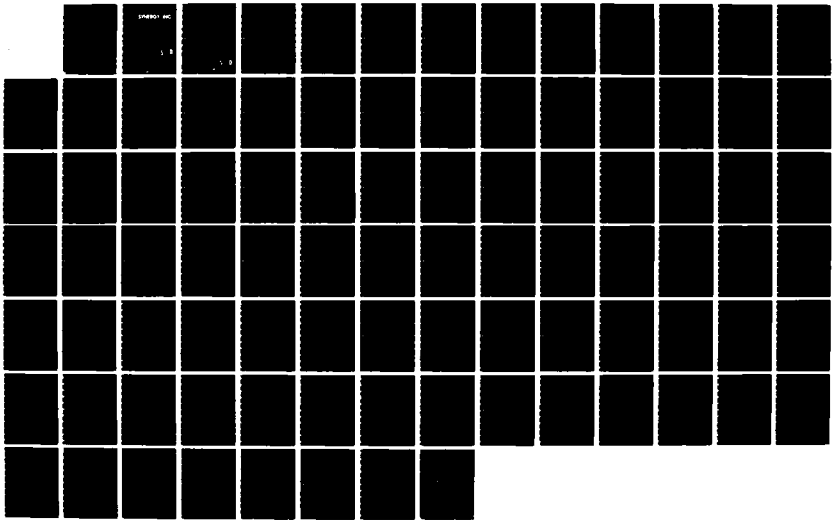
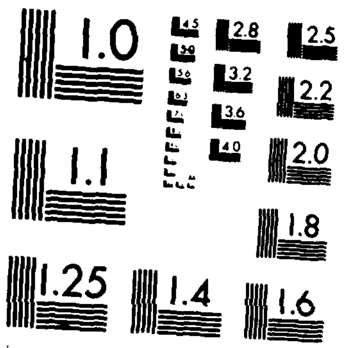
AD-A173 697 AN EXPERT SYSTEM TO HELP ASSESS TACTICAL AIR READINESS AND CAPABILITY PHASE I REPORT(U) SYNERGY INC WASHINGTON DC 30 MAY 86 F49642-85-D-0035 1/1

UNCLASSIFIED F/G 15/7 NL




1.0
4.5
2.8
2.5
5.0
5.6
3.2
2.2
6.3
3.6
4.0
1.1
2.0
1.8
1.25
1.4
1.6

AN EXPERT SYSTEM TO HELP ASSESS TACTICAL AIR READINESS AND CAPABILITY

PHASE I REPORT

Contract F49642-85-D-0035/5002

1824 Biltmore Street, NW
Washington, DC 20009

(1)

AN EXPERT SYSTEM TO HELP ASSESS TACTICAL AIR READINESS AND CAPABILITY

PHASE I REPORT

MAY 30, 1986

Prepared by Synergy Corporation and Systems Research and Applications Corporation (SRA) for the Directorate of Operation, Readiness Assessment Group (HQ USAF/XOOIM) under contract F49642-85-D0035/5002.

TABLE OF CONTENTS

Executive Summary

I. Introduction

[illegible]

~~We are~~ building a demonstration expert system that allows users to ask questions in English about major resources and their effects on U.S. tactical aircraft sorties in central Europe. The system will handle two types of questions. The first are those that require "understanding" the request, knowing what data bases (if any) to search, searching those data bases, selecting the right piece of information, and presenting the answer to a user in a format he wants. The second type are those questions that require substantive expertise and thinking to answer (i.e., those that require both an "intelligent" search for information and expert analysis of that information).

The demonstration system will answer questions drawing on 6 types of information:

- Characteristics of U.S. tactical aircraft in Europe;
- The specific missions those aircrafts fly;
- The number and type of conventional munitions associated with U.S. tactical aircraft;
- Availability of and POL requirements for U.S. tactical aircraft;
- Aircrew status and availability; and
- Status and descriptions of 4 U.S. tactical airfields (Spangdahlem, Bitburg, Hahn, Ramstein).

Not only will the system know facts (e.g., be able to answer "how much JP-4 do I have at Spangdahlem?"), but also think about more complex questions that require expert knowledge (e.g., "How many sorties will that JP-4 support?"). In addition, the demonstration system will answer excursion or "what if" questions (e.g., "If I lose 1 runway at Ramstein and 50% of my POL, how many sorties can I get from my F-4s in the next 5 days?").

The system will answer the user in the format he wants. If the user wants graphs, he will ask for and get graphs; if he wants answers in English, he'll get answers in English; if he wants both, he'll get both.

The system will also keep a "logic audit trail" for each answer. If a user wants to know why the expert system gave an answer, the system will explain in English the logic it used to get the answer. This logic will include any assumptions the system made, the data it used, and the analyses performed on the data.

The system will recognize ambiguities in a question. For example, if a user asked 3 separate questions about the number of aircraft at 3 different bases in Europe (e.g., "How many aircraft do I have at Bitburg?" "How many aircraft do I have at Hahn?" "How many aircraft do I have at Ramstein?"), and then asked "How many sorties will they fly in the next 2 days?", the system will recognize the ambiguity and ask the user whether the sorties are for aircraft at all 3 bases or just those at Ramstein.

When there is not enough information in a question to answer it, the system will "know" what additional information is necessary and ask the user for that information. For example, if a user asks "If I lose an airfield, how will it affect my capability?", the system will ask for additional information: "What airfield would you lose, and for how long would you lose it?"

II. A Brief Description

The best way to show what our expert system will do is to show what a dialogue with it would look like. The system will allow a user to sit at a terminal, type in questions about our tactical air capability in Europe, and automatically generate the answers. The user might begin the dialogue with a question such as: "How many F-4Es are MRA in the central region?" The system would provide an answer in English. The answer might be "80 F-4Es are MRA in the central region." The system understands the meaning of the question and the meaning of specialized words in the question. For example, the system knows that "MRA" means "mission ready able."

The user could have asked the same question in many different ways. The system breaks a sentence down into a "deep syntactic" structure that represents a common meaning for different "surface structures." Therefore, given reasonably different ways of asking a question, if the meaning is the same, the deep structure will be the same. The system responds in the type of English the user understands.

The dialogue might continue with a second question such as: "How many of those are at Spangdahlem?" The system responds with an answer such as: "42 of the F-4Es are at Spangdahlem." In this example, the system is "smart" enough to "know" that the word "those" refers to F-4Es. The user does not have to repeat "F-4Es." In response to this question, the system determines its meaning, decides what information it needs, locates where that information resides, and retrieves it.

The dialogue might continue with a third question such as: "How many sorties were those supposed to fly in the first 10 days?" In this example, the system once again knows that the word "those" refers to F-4Es. The system would respond with two answers. The first would be: "The F-4Es at Spangdahlem were supposed to fly 10 sorties in the first 10 days." The second answer would be: "The F-4Es in the central region were supposed to fly 1,000 sorties in the first 10 days." The system gave two answers because the question was ambiguous. The question could have referred to F-4Es either in the central region or those at Spangdahlem.

At this point, the expert system was faced with several choices. First, should it respond to both potential meanings of the question? Second, should it respond only to the most recent reference in the question? Or, third, should it ask the user to resolve the ambiguity in the question: "Do you mean F-4Es in the central region or F-4Es at Spangdahlem?" In the example, the system answers both implied meanings of the question.

The previous dialogue uses an expert system to understand the meaning of a query, to identify what facts or information it needs, and to find that information. There are other, more complicated questions than an expert system would handle. These questions require not only understanding the question itself, but also understanding how to set up an analysis to answer the question. The analysis must not only "know" what information it needs, but also how to analyze that information. For example, a user might continue with our sample system by asking a question such as: "What is the largest number of sorties we can get from the Spangdahlem F-4Es in the first 5 days?" The system might respond by saying: "You could get 50 sorties the first 5 days." The expert system calculated the answer based on the "meaning" of the question, the information available, and the analysis it performed.

The dialogue might continue with another question such as: "Why can't we get more?" In this case, the system understands that "more" again refers to sorties, and not to aircraft or airbases. In responding, the expert system might say: "Because there is only enough JP4 for 50 sorties." Here, the system introduces a variable or a limit that the user did not introduce -- JP4, aviation fuel.

In continuing the dialogue, the user might ask: "What if we have enough fuel?" Here the system understands that the implied objective is the number of sorties that might be generated by the F-4Es at Spangdahlem. That implied objective does not have to be repeated, because the system keeps it "in mind." The system responds with: "350 in the first 5 days." Here the system reasons that, if there were no limitation in fuel, far more sorties could be generated. If the user had asked what the next limitation would be, the system would have responded with an answer: munitions, crews, the number of aircraft available, etc.

Changing the subject of the dialogue, the user might ask a different question, such as: "If we lose one runway at Spangdahlem, how would it affect the total number of sorties for everything we've got in the central region?" Here, the system might respond by saying: "The total number of sorties would stay the same but it would take you longer to fly them. The time depends on how long the runway at Spangdahlem was lost." In this example, the expert system knows how to do "what if" excursions. The system can answer hypothetical questions dealing with contingencies that have not yet occurred.

In answering the question about a lost runway, the system understands that the basic objective is to maintain high sortie rates. It analyzes the consequence of losing a runway and determines that the objective of high sorties can still be met but over a longer time. The expert system responds appropriately.

The system understands this complex dialogue, in part, because it has a basic goal -- keeping sorties high -- that helps shape both its understanding of the questions and the analyses it must perform.

The user ends the dialogue by asking a simple question: "Why?" The "why" refers to the previous system response that the total number of sorties would stay the same but take longer. The system answers this "why" by providing the logic that led to the answer. In other words, the system tells the user what assumptions it made, what information it used, how it used that information, the calculations it made, and the conclusions it came to. This is a logic audit trail, a trail that tells the user why the system concluded what it did.

III. How the Demonstration System Will Work

There are four major components of the demonstration expert system that we propose to build. Figure ES-1 shows these components. The first is the component that structures the question (i.e., it breaks the question down into its deep syntactic structure). The second module understands the meaning of the question. It does so by relying on a "knowledge base" and a series of semantic labels that describe the characteristics of words in the syntactic structure. In essence, the second module gives meaning to the syntactic nodes produced by the first module. The third module designs the logic of the search, searches for appropriate information, and then analyzes the information to determine an answer. The fourth module generates the answer. In some respects, the fourth module is an analog of the first module. The first module goes from an English expression to a deep syntactic structure. The last module goes from a syntactic structure to an English expression.

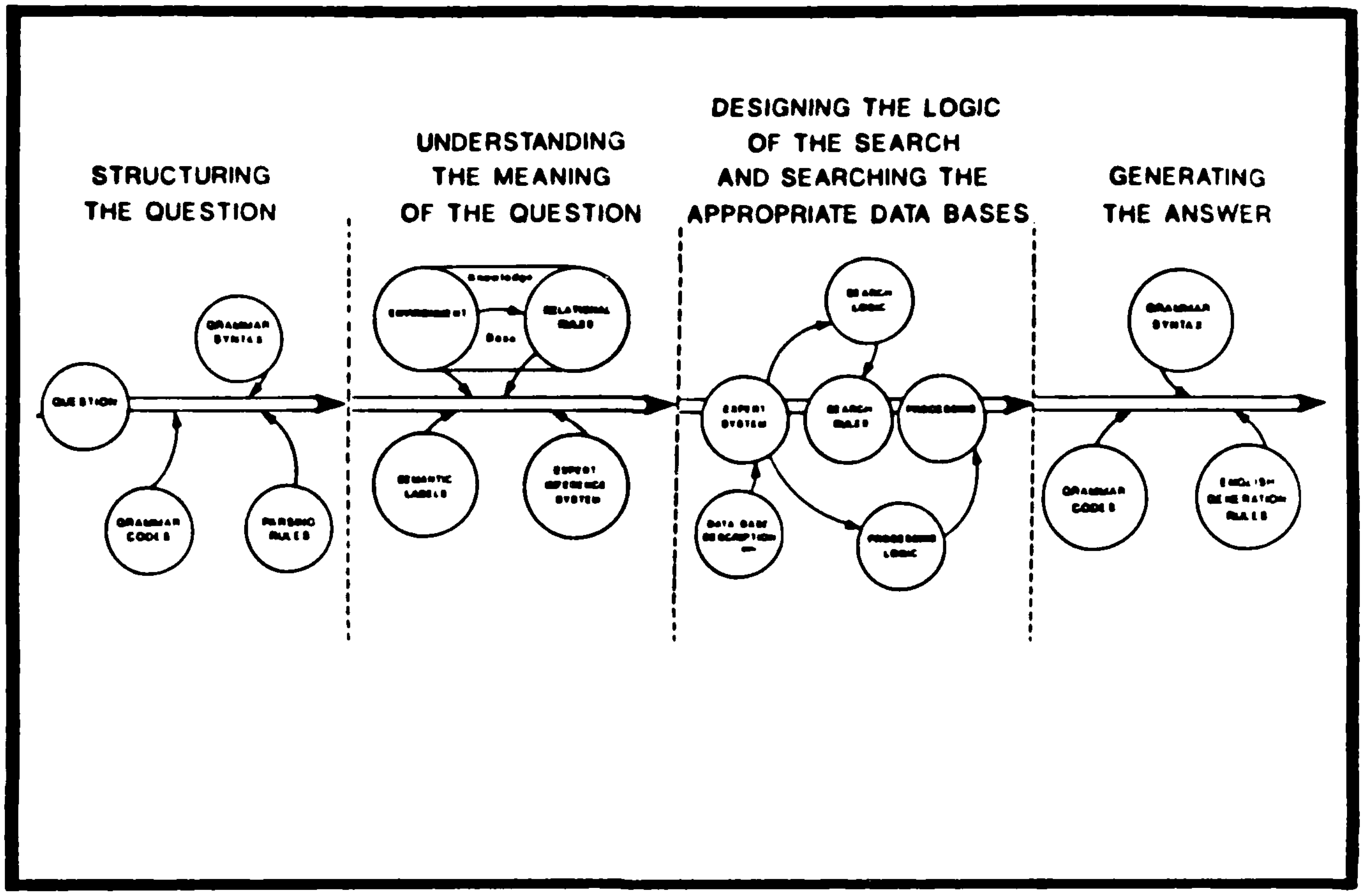


Figure ES-1. The Parts of the Tac Air Expert System

I. INTRODUCTION

1.1 The Nature of Artificial Intelligence

"Artificial Intelligence is the study of ideas that enable computers to be intelligent."

"Artificial Intelligence (AI) is the part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit the characteristics we associate with intelligence in human behavior -- understanding language, learning, reasoning, solving problems, and so on."

There is some confusion about what artificial intelligence is. We start by saying what it isn't. Artificial intelligence is not sophisticated programming. Sophisticated programming can be very complex and powerful. But complexity and power are not necessarily the only attributes of artificial intelligence. There are two important characteristics of AI that reflect human intelligence and distinguish AI from conventional programming. The first characteristic is that AI demonstrates the capability to understand language, learn, reason logically, or solve some problems requiring "intelligence". The second characteristic of artificial intelligence is that it performs an intelligent funct' , in a non-trivial way; that is, the performance represents the way an intelligent person might approach a problem. These two characteristics differentiate artificial intelligence from conventional programming.

As Figure 1-1 suggests, artificial intelligence includes both hardware and software. Although there are several different aspects to the software domain, natural language processors and expert systems are two important parts. Natural language processors translate English sentences and phrases into a "deep structure" that represents the funda-

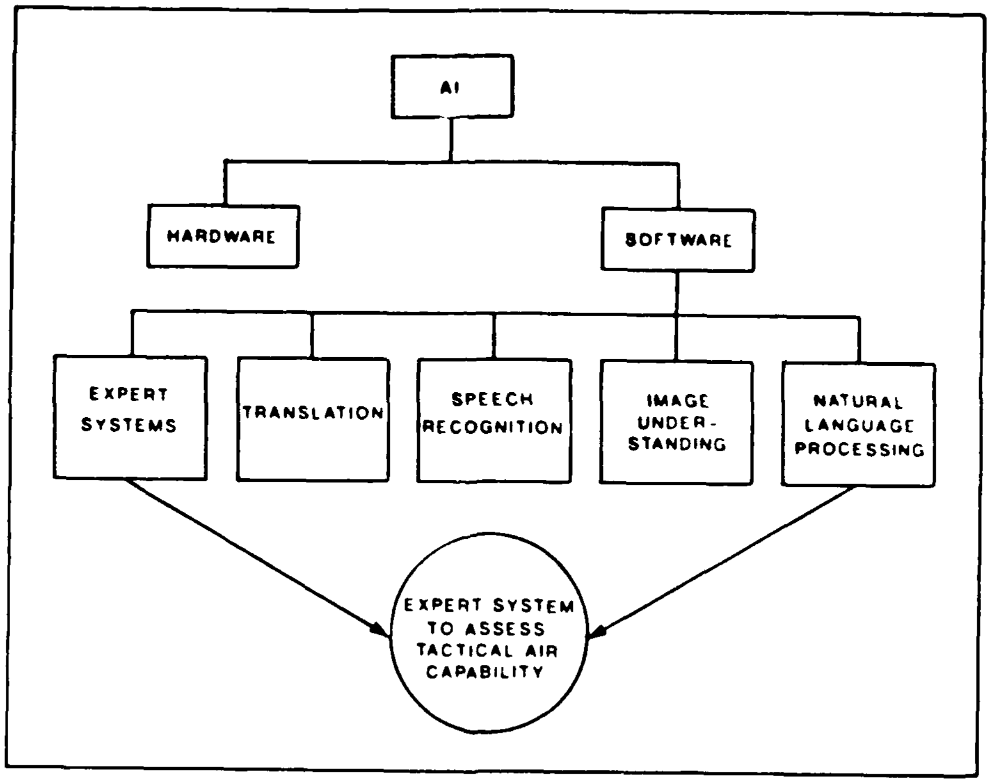


Figure 1-1. The Parts of Artificial Intelligence

mental meaning of the sentence or phrase. An expert system is "... a computer system that encapsulates specialized knowledge about a particular domain of expertise and is capable of making intelligent decisions within that domain." In essence, an expert system captures both the knowledge and logic a real expert uses to solve a problem. We propose to combine a natural language processor and an expert system so that a user can communicate with the system almost as easily as he could communicate with a live expert.

1.2 The Value of Expert Systems

Why do we need expert systems? There are several answers. First, expert systems are always available. Sometimes, when expertise is needed, the person who has the expertise may not be available. He may have been transferred; he may be sick; he may be on leave or TDY. There may be a number of reasons why the expert -- and his expertise -- is not available when and where we need him. An automated expert system is always available.

Second, an expert system is consistent. By consistent, we mean the system handles the same problem in the same way. This may be unlike people who look at the same problem differently depending, for example, on whether they have just studied the problem or not.

Third, an automated expert system is systematic. By systematic, we mean that it keeps an audit trail of how it reaches a conclusion or does an analysis. The expert system's audit trail includes the assumptions the system makes, the information the system uses, where it got the information, and how it analyzes the information.

Fourth, expert systems are fast. They can operate almost as if you were dealing with a human expert. For more complex problems, the expert system can provide a quicker response than the human expert because the expert system has access to all the information necessary to answer a question.

Fifth, expert systems can be improved. Once an expert system is designed, it can be tested to see where it is weak and then strengthened in those areas. Because it is systematic and consistent, it can be systematically and consistently improved, as long as we have the experts with the expertise to improve the system.

Finally, expert systems have the potential for self-improvement. We say potential because there is no system yet that can learn, in the human sense, from its own operation. Perhaps ten years in the future, there will be systems that, based on their simple operation, will change their rules and add to their knowledge base; but as of today, the only systems that improve themselves do so in trivial ways.

1.3 The Expert System vs. Traditional Approaches

Having said something about the value of expert systems, we would like to contrast the way expert systems work with traditional approaches to automated support. In a standard approach to information searching (Figure 1-2), there are generally four stages. The first stage is a query stage. A user formulates a question, decides what information is necessary to answer the question, and identifies the sources of that information. In the second stage, the user retrieves the information that is necessary based on his design. The third stage is an assessment stage in which the user takes the information that he retrieved, and analyzes, reconfigures, and manipulates it to get the answer to the query he first posed. He uses the automated support to search the available data bases.

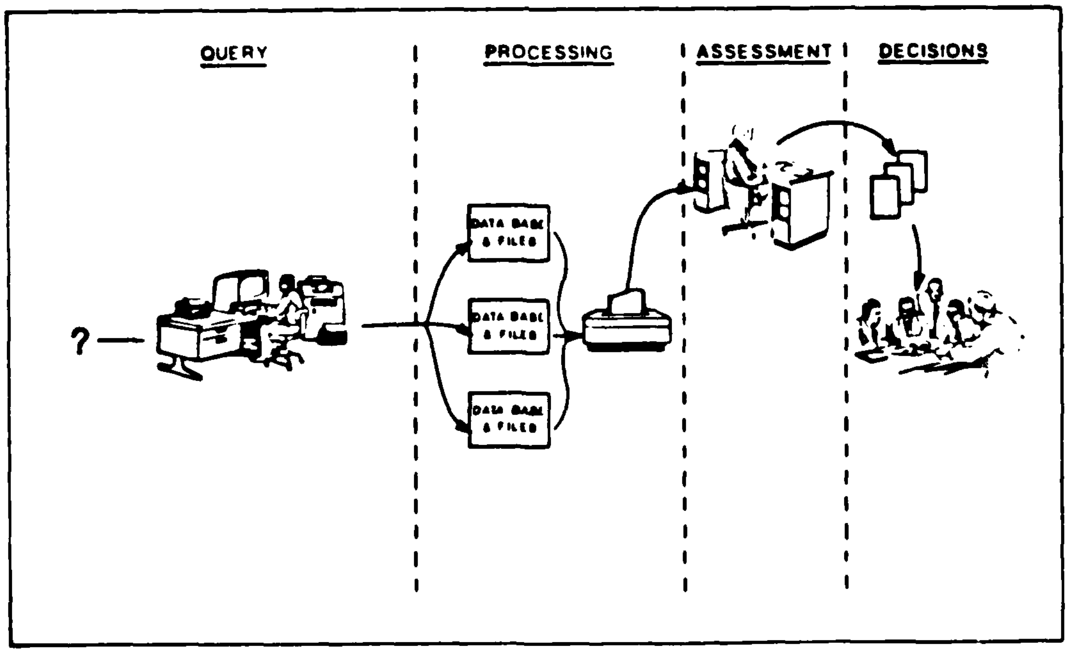


Figure 1-2. Standard Approach to Information Searches

An expert system takes a different approach to automated support (Figure 1-3). An expert system anticipates the environment (i.e., the general area in which queries might take place). The expert system does not anticipate specific questions but rather the environment in which

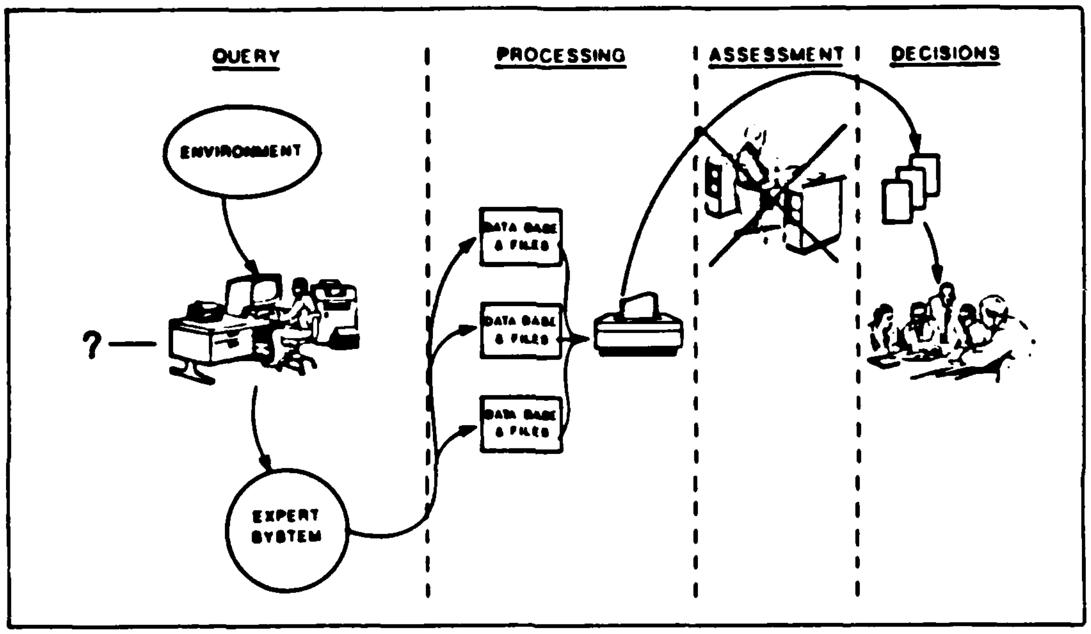


Figure 1-3. AI Approach to Information Searches

those questions fall. The expert system also knows what information exists in what data bases, and how to translate a specific query from a human user into a research design. It takes a human query, structures the analysis necessary to answer that query, searches the appropriate data bases, and then performs the actual analysis. In essence, the expert system has replaced the human expert in structuring the question, and retrieving and analyzing the appropriate information.

1.4 Definition of Terms

Expert systems are distinguished from other software systems in several ways:

1. The data processed by an expert system is the knowledge of how to solve a problem. This knowledge is directly available to the user, not hidden in the program code.

2. Most information is processed "symbolically" instead of "numerically". In expert system applications, problems are approached in high-level terms appropriate to the particular application. Artificial intelligence languages (such as LISP, PROLOG, and higher-level tools) facilitate this view of problems, both for the developer and the customer.

3. Expert systems can usually explain their answers in terms of the logic (expertise) used to solve the problem.

4. Since expert knowledge is the data processed by expert systems, it can be enhanced or corrected as easily as numerical data can be changed in conventional systems. This means that the programmer (sometimes even the user) may change the approach to solving a problem without changing the programming. This is why prototype expert systems are easy to develop and improve.

As suggested earlier, an expert system has a number of major components:

KNOWLEDGE BASE. A knowledge base is the collection of rules, frames, and other data that the expert system uses to reason. The terms "rule" and "frame" are described below. The knowledge base is typically a combination of the expert's knowledge and higher-level problem solving knowledge. This higher-level knowledge is known as "meta-knowledge," or "knowledge about knowledge." Meta-knowledge serves two purposes: (1) it allows the expert system to reason at a higher level, improving the efficiency of the system; and (2) it allows the expert system to understand itself, enabling it to explain its reasoning process.

FRAME. A frame is a data structure describing an "object" in the world of an expert system. Objects are any thing that deserve to be considered as individuals, and frames are the symbolic representations of those objects. Frames contain "slots," which name the important attributes or characteristics of the associated objects. Slots may have one or more values, usually either pure data (symbols or numbers) or the names of other frames, in the knowledge base. Slots containing the names of other frames provide the capability to link frames, establishing relationships between objects that the expert system may use in solving a problem.

RULE. Rules are the direct representation of a fragment of an expert's approach to solving a problem. Expert systems derive their power from the power of the "heuristics," or "rules of thumb," that the expert uses to quickly solve problems. Meta-knowledge also takes the form of rules. Look at rules as the general form: IF (CONDITIONS) THEN (CONCLUSIONS); if each CONDITION in a rule is true, then each CONCLUSION can be considered true.

Any rule can be viewed from two directions. For a given rule, proving all of the conditions true allows the expert system to assert a rule's conclusions. Conversely, a question can be answered by finding a rule with the answer to the question as part of the rule's conclusions, thereby proving all of the rule's conditions. The first view looks at the rule in the "forward" direction, and is useful for responding to input data or changing situations. The second view looks at the rules in the "backward" direction, and is useful in performing diagnosis or responding to queries.

SEMANTIC NETWORK. The world of the expert system is viewed as a collection of objects. Objects are linked together semantically through "relationships." As an example, the CAR object might be linked to the VEHICLE object by a relationship called "is a kind of," "is a subclass of," etc. For illustration, think of the link as (IS A), as in "A CAR IS A VEHICLE". The same relationship holds between an APPLE object and a FRUIT object, or between an F-15 object and an AIRCRAFT object. The notion of "inheritance" allows the expert system to follow these links to more general conclusions. The CORVETTE object (IS A) CAR, and a CAR (IS A) VEHICLE, therefore the expert system knows (and can use) the fact that a CORVETTE (IS A) VEHICLE, even though the two objects are not directly connected.

Large semantic networks have hundreds or thousands of objects loosely joined together by several kinds of relationships; the particular relationships connecting objects depend on the application of the semantic network. Objects in semantic networks are usually represented as frames. Some slots of the frames are true characteristics, such as NUMBER-OF-WHEELS, MILEAGE, and NUMBER-OF-SEATS. Other slots are relationships, such as the (IS A) slot for linking subclasses with classes with superclasses of objects, or the (PURPOSE) slot for linking a CAR object with another object dealing with activities. For the CORVETTE object, the (PURPOSE) slot might be CRUISING, while the CHEVETTE object's (PURPOSE) slot might be COMMUTING and SHOPPING. CRUISING, COMMUTING and SHOPPING would be other frames (objects) in the semantic network.

The (IS A) link between objects facilitates the inheritance of characteristics from higher-level (more general) objects. For example, since all cars have four wheels, the NUMBER-OF-WHEELS slot would best reside in the CAR object, with value 4. All objects subordinate to CAR -- CORVETTE, MUSTANG, CHEVETTE, etc. -- would inherit that value from CAR. Thus, if the expert system needs to know how many wheels a COR-

VETTE has, it is indirectly available by following (IS A) links. This notion of inheritance allows a system to be designed modularly and efficiently. Information resides in the most general and appropriate location and is shared through inheritance.

INFERENCE ENGINE. The inference engine is the brain of an expert system. It is the software that uses rules in the knowledge base to respond to input data or answer questions. As mentioned earlier, rules work in two directions, forward and backward. In the forward direction, the assertion of a rule's conclusions may cause another rule's conditions to become true. That new rule's conclusions would then be asserted, possibly causing yet more rules to "fire." This spreading activation of rules is known as "chaining," and it works in the backward direction as well.

In backward chaining, a rule is selected for processing because its conclusions contain the desired answer to the question at hand. For the conclusions to be proven true, all of the rule's conditions must be shown to be true. Proving one of the rule's conditions may require another rule to be activated in the backward direction, requiring another set of conditions to be proven valid. In both forward and backward chaining, it is important for the inference engine to select the order in which to process the activated rules. Both parts of the selection process, known as "matching" and "conflict resolution" are very important to the performance of an expert system.

NATURAL LANGUAGE PROCESSOR. Most expert systems have an English-like user interface allowing data input or questions in a somewhat uncontrained form. Data base management systems often have similar interfaces, allowing the users to query the data base using simple English questions. True natural language processing is far more complex than a data base or knowledge base query language. It must handle free-form English text, ambiguity, spelling and grammatical errors, and memory of recent context. All of those problems are routine for humans, yet are very difficult to handle by computer. Most natural language programs have a detailed knowledge of grammar, a large vocabulary, and an extensive knowledge base (usually a semantic network) that allows the natural language system to understand the statements and questions. The following paragraphs discuss the most difficult problems handled by sophisticated natural language systems.

CONTEXT. Humans retain memory of recent conversation, and are able to "fill in the blanks" of any statements or questions that depend on the preceeding dialog. In order to understand questions such as "How many of those are red?" or statements such as "Take the smallest and divide it by ten," natural language systems must remember the context of the discussion and determine the actual referents of "those," "smallest," and "it."

AMBIGUITY. Even humans have trouble with ambiguous sentences, shown by the necessity to think about or ask further questions about an ambiguous sentence in order to understand it. There are two types of ambiguities in natural language systems that handle English text: syntactic and semantic. (In systems that understand spoken language, there is a third -- the trouble that homonyms, such as "blue" and "blew," cause). Syntactic ambiguity arises from words that act as two or more parts of speech. In the sentence "He saw that gasoline can explode," there is some syntactic ambiguity: "can" could be viewed either as a verb or a noun, and either way would be correct grammatically. The natural language processor must determine the most likely use of "can," resulting in an interpretation of either "He saw a particular can of gasoline blow up," or "he saw that gasoline has the ability to blow up." In the sentence "He saw the man in the park with his telescope," there is no syntactic ambiguity, but considerable semantic ambiguity. The system must determine:

1. Who was in the park?
2. Whose telescope was it?
3. Who had the telescope?

Reference to the recent context of a conversation is necessary to understand even simple ambiguous sentences.

GRAMMAR/SPELLING. Humans may not like dealing with grammar and spelling errors, but they cope with such errors easily, unless the errors result in ambiguity. Ungrammatical conversations are fragmentary, but often very efficient. We have no trouble responding to questions as cryptic as "Ramstein status?" as long as the question is within an appropriate context. Natural language systems must ignore spelling errors to some extent and simple grammatical errors must not cause the system to balk.

II. Scope of Effort: The Domain of the System

AI and expert systems have been most successful when applied to discrete, well-defined, and well-understood areas. Earlier attempts to develop "general problem solvers" that could successfully deal with any problem (from theorem proving to military planning) failed. In general, the more one knows about a smaller area, the better the expert he is. Our demonstration expert system is expert enough in a large enough domain to be of interest and value to the Air Force, but not so large that it requires the knowledge of the universe to operate.

2.1 Capabilities

The expert system will understand typed English sentences and will respond in English as well as selected graphic displays. The system will understand operations involving the F15, F16, F4E, and F4G aircraft based at Bitburg, Hahn, Ramstein, and Spangdahlem airbases. Daily, crisis, and wartime conditions are covered. The system will have access to knowledge concerning these bases normally available at the Air Staff level. This knowledge will be stored in a knowledge base. Table 2-1 defines the seven factors that define the scope of our efforts.

A useful way to think about the system's capabilities is to imagine an artificial world created by drawing a line around the four airbases and viewing the result from the perspective of the Pentagon. The system doesn't know anything outside the region containing these airbases. For example, it does not know about resupply of POL, aircraft, aircrews, munitions, or spare parts from outside the region. The system's world starts with the amount of these commodities that are really there. The Air Staff Officer using the system can tell it to assume that any desired amount of these commodities is present. Thus he could achieve

Table 2-1

Defining the Scope of Effort

- REGION - EUROPE: THE CENTRAL REGION
- AIRFIELDS - FOUR U.S. TACTICAL AIRFIELDS (BITBURG, HAHN, RAMSTEIN, SPANGDAHLEM)
- AIRCRAFT - U.S. TACTICAL AIR
- MISSIONS - NON-NUCLEAR TACTICAL
- SUPPORT - MEN, MATERIEL
- EFFECTIVENESS - "STANDARD" SORTIE
- ORGANIZATION LEVEL - HQ USAF

the effect of adequate resupply of POL by saying: "Assume we can supply enough fuel at all bases," then all subsequent problems would be worked out as though there were always enough fuel.

Aircraft, when they fly a mission, leave the system's world. Consequently, in the absence of any other information, it assumes standard planning factors for mission duration, fuel consumption, munitions expenditure, attrition, and malfunctions. The Air Staff Officer can make up for the system's lack of mission planning capability by changing the standard factors for certain missions. For example, he might say: "Assume full expenditure of all missiles on DCA sorties."

Because the system's viewpoint is that of the Pentagon, it knows that aircraft are launched, recovered, refueled, rearmed, repaired and parked at an airbase. The system will not, however, know all the information of a wing commander (e.g., the current status of aircraft by tail number). It will know who has that information, however. Without any other guidance from the user, the system will make its own best judgment on what standard factors it should use for recovery, refueling, and rearming aircraft that do not need repairs. However, like a real expert, the system can make different assumptions if the user wants it to. Again, the Air Staff Officer could state: "Assume 1.5 hours for turnaround," and thus achieve the effect of bad weather conditions in the area. If a question is too detailed for the system to answer, such as "What specific aircraft, including wargoers, are available for air defense at Bitburg?," the system would answer: "There are 58 F15s available for air defense at Bitburg, including 8 wargoers. Only the wing commander knows the specific aircraft."

2.2 Comprehension

To suggest how the system understands what the Air Staff Officer is saying, we first define a few terms.

SESSION. Everything that takes place from the time an Air Staff Officer starts to use the system until another Officer uses it or the system is turned off.

PROBLEM CONTEXT. The set of facts that are known about the problem being discussed.

SITUATION. The set of facts that are known about the environment in the region of the four airbases.

DEFAULT SITUATION. A set of facts the system assumes about the environment in the region of the four airbases when nothing else is known.

Suppose the system is asked the question: "How many sorties can I generate?" The system knows what procedures to use to answer the question. Whenever a procedure requires information about the problem, the system first looks for the information in the question itself. If it is not available there, it looks for information in the current problem context. If it is not available there, the system looks for the information in the current situation. If it is still not available there, it looks for the answer in the default situation. For example, the question does not specify what airbases are to be used to generate sorties. If the previous sentence had been: "Suppose Hahn has 5,000,000 gallons of fuel," the problem context would be limited to Hahn airbase. If the previous sentence were: "Let's work a new problem," the problem context would be empty and the current situation would be examined. If the current situation were a surge exercise at Hahn and all other airbases were in a normal daily state, the problem would again be interpreted as referring only to Hahn airbase. If no environment had yet been established in the current session, the default environment, which is the first day of the invasion of Europe, would apply and all airbases would be used. In general, the system replies based on the best assumptions it can make and lists these assumptions in the reply. This is different from asking for clarification of each point; this approach gives the Officer the opportunity to say: "No. I meant all airbases, not just Hahn" and get a new answer.

The problem context and situation are useful concepts that allow the Air Staff Officer using the system to ask his questions briefly. This alleviates the problem often encountered in English language systems of being too wordy and slow to use when compared to menu or command driven systems. We have elected to keep the problem context and the environment constant throughout a session until canceled by something like: "Now let's look at a new situation" or until modified by something like: "Apply the same conditions to Ramstein, how many sorties can I generate there?" In addition, some simulated information in Air Force documents, such as WMP-5 and TAMP, are available to provide default values for many factors that affect the solution of a problem. This avoids the need to specify these factors unless the Officer wishes to use his own. Also, it may be possible to override these factors wholesale instead of one at a time by saying something like: "Use OPLAN 4496" or "Use the standing ATO" which would cause the more specific information in these documents to replace the corresponding default information from the WMP. In the demonstration system, we will show this capability with simulated parts of the ATO, WMP, and OPLAN.

2.3 Knowledge Base

[illegible]

The knowledge base stores all the facts and information the system knows. Some of the facts can be obtained from existing documents, such as the capabilities of aircraft. However, there is lots of knowledge about how and when to use information and about Air Force capability assessment that cannot be found in documents. The method used to determine this portion of the knowledge base is best described as an inductive prototyping approach. [illegible] ? An expert in Tac Air assessment discussed typical problems with us. We recorded and transcribed these sessions. We began "from the bottom up" by using an unclassified version of "Weeping Willow" -- an operational plan for maintaining free air corridors into Berlin. Based on the expert's description of this plan, we began to build the "objects" in the knowledge base (i.e., to "teach" the system about its world). As more and more problems are added, the knowledge base will evolve towards its final form.

The rest of this sub-section describes how we could use this problem to help build a knowledge base. It is an example of how we might capture some operational knowledge that could help answer more general resource allocation problems. Though our demonstration system will not be an operational planning expert, we use the example to highlight the knowledge base building process. In the initial interviews, we discovered two types of problems. The first type, represented by the question, "How long a runway does the F4E require to land?" can be answered by looking up a fact in the knowledge base. A lot of reasoning is required to understand the question, but little reasoning is required to answer the question. The second type of problem is represented by the question: How long can I continue to support phase II of a certain operational plan? This question is answered by constructing a mental model of the operation and carrying it out until some resource is exhausted. The answer cannot be looked up in the knowledge base. A system must carry out the reasoning necessary to construct the model and find the answer. As examples of the type of reasoning required, three problems will be discussed here: the "do something specific" problem, the "recommend what to do" problem, and the "how much can I do" problem.

Task 2D

The "do something specific" problem is exemplified by Phase I of the Weeping Willow operational plan. To demonstrate our resolve to keep the three air corridors to Berlin open, the plan calls for the establishment of a suppression of enemy air defense (SEAD) cap protected by a defensive counter air (DCA) cap at the western end of each corridor. The plan tasks Spangdahlem airbase to provide the F4E and F4G pairs for the SEAD caps and Bitburg airbase to provide the F15 pairs for the DCA caps. Typical questions about this operation that could be posed to the Air Staff would be:

"Can we support Phase I of Weeping Willow? How long? Are there any showstoppers? What do I have left?"

To answer such questions, a system would use its knowledge of the operational plan, its knowledge of SEAD and DCA mission types, its general knowledge of aircraft and theater, and its general problem solving ability to determine the number of aircraft required. A system can know that airbases are designed to support full utilization of their aircraft. So if, as in this case, the required aircraft were less than airbase capacity, a system would look at readiness data for aircraft, crews, POL, armament, and maintenance factors to see if the percentage available for each factor exceeds the percentage of aircraft required. Thus, a system might answer that Phase I could be supported indefinitely assuming normal rates of resupply. In addition, a graphic depiction of the available aircraft would indicate the proportion of committed versus available resources.

The "recommend what to do" problem occurs when an unarmed aircraft is shot down in one of the air corridors. This is covered in phase III of the Weeping Willow operational plan, but the plan is not specific about what to do because the specific situation could not be foreseen. In this situation, the Air Staff might ask a question like: What kind of retalitory attack do you recommend?" If a system had strike force planning knowledge, it might reason out the answer. A system might know that the objective is to destroy the SAM site that shot down our aircraft. Secondary objectives are to minimize escalation of hostilities since we are not yet at war and to maximize the chances of mission success without losses. Intelligence estimates ten SAM sites in the region of the target site. Two F16s give a high probability of kill but also have a moderate probability of being detected so protection is required. Two SEAD flights could cope with the expected level of SAM threat. Each SEAD flight should have a DCA cap for protection against air threats. Precision munitions should be used to minimize collateral damage. An "ultimate system" might suggest a sweep of two SEAD F4s followed by 2 DCA F15s, 2 airground F16s, two more DCA F15s, and another 2 SEAD F4s at 3 mile intervals. In addition, an operational planning system could recommend re-roling two of the F16 squadrons at Hahn to air-air alert to minimize the offensive nature of our posture. The remaining F16 squadron can supply the air-ground F16s for the strike mission. The F15s come from Bitburg and the F4s come from Spangdahlem because those are the only sources of these aircraft. A complete system would use an analysis similar to that pefomed for the first problem to make sure this recommendation is feasible as well as desirable.

The "how much can I do" problem occurs in Phase IV of Weeping Willow when the Soviets decide to attack our caps with air and ground forces. The Air Staff is asked:

"How many fully armed sweeps per day can I fly to keep the air corridors open? What are the limiting factors?"

Again, a complete planning system would use its force planning knowledge to determine that each sweep mission will consist of two F16s with half air-air and half air-ground munitions and a SEAD pair using an F4E and F4G. One important rule in this situation is that to maximize sweeps, it must minimize aircraft used per sweep. A pair of SEAD F4s is required to suppress the longer range SAM sites. A pair of F16s or F15s are required to counter the air threat to the F4s. In this case, F16s are chosen since they are more numerous, less expensive, almost as good at air-to-air combat as the F15s, and can be armed with some air-ground munitions to counter visual range SAM threats. These sweeps will be riskier than the retaliatory attack mounted in Phase III with ten aircraft. This is justified by the need to maximize sweeps. After developing the aircraft requirements of each sweep mission, a real expert derives crew, POL, armament, and maintenance requirements on a per mission basis. Using the remaining resources, this information would enable a complete system to calculate the maximum missions per day and report on the first five limiting factors. In this example, F4s limit missions on the first day, the available AIM-9 missiles at Hahn limit missions on the third day, and POL becomes a problem at Spangdahlem on day 4.

Some of this detailed knowledge could clearly help our demonstration system understand and think about more general capability assessment issues.

2.4 Content

Although the complete content of the knowledge base has not yet been determined, here is an example of some of the rules it might contain based on the foregoing analysis. Keep in mind that the knowledge would probably not be represented this way in the real system. This example is for illustrative purposes only.

Rule: If situation is show-of-force, then hostile attrition factor is zero.

Rule: If aircraft is F15 or F16, then mechanical attrition factor is 10%.

Rule: If aircraft is F4, then mechanical attrition factor is 20%.

Rule: Attrition factor is mechanical attrition plus hostile attrition factor.

Rule: The endurance of an F4 is 1.7 hours.

Rule: The turnaround time of an F4 is 1 hour.

Rule: The average time spent on station in a SEAD mission is .9 hours.

Rule: A SEAD mission requires an F4G and an F4E.

Rule: If continuous manning is required, the number of aircraft required is $N * [\ (E+T)/D\ ] * A$, where:

- N is the number of aircraft required per mission.
- E is the aircraft endurance
- T is the aircraft turnaround time
- D is the time spent on station
- A is the attrition factor

Collectively, these rules enable the system to decide on how many aircraft are required to man a SEAD cap continuously.

2.5 Structure

The knowledge base contains a wide variety of information. That information can be classified in the following way.

Situation
: The knowledge about the conditions surrounding the problem, e.g., are we at war or not?

Discourse
: The knowledge about the topic that we have been discussing, e.g., the subject has been Weeping Willow.

Scenarios
: The knowledge of likely sequences of events, e.g., if we miss the target, we may hit civilians and force our opponent into even more intense retaliatory attacks.

Syntax and Semantics
: The knowledge needed to understand English, e.g., F15 is a noun and F15 is an aircraft.

Meta-knowledge
: The knowledge about when or how to use knowledge, e.g., if aircraft requirements are not specified for a mission, use force planning knowledge to determine them.

Air Force knowledge
- Plans
  - e.g., Weeping Willow,
- Resources
  - e.g., aircraft performance data,
- Procedures
  - e.g., a SEAD mission requires an F4G hunter and an F4E killer.

The next step in the construction of the system is to encode this knowledge in an efficient form.

III. Natural Language Processing

Task 2E

As we suggested earlier, an important part of our demonstration system will be a "natural language (NL) parser" -- a software program that "understands" the meaning of a question.

The purpose of a natural language parser is to determine the "deep syntactic structure" of a sentence. "Syntactic structure" is a way of representing the parts of a sentence and their relationships to one another. For example, labeling parts of a sentence as "actor items," "action items," and "object items" is one way of defining the "syntactic structure" of a sentence or phrase. Consider the following sentence:

"F-16s at Hahn flew 200 sorties against massed Soviet armored divisions in the Fulda Gap."

This sentence could be "processed" and then labeled using our simple categories.* The result would be the following:

Actor Items	[F-16s at Hahn]
Action Items	[flew 200 sorties against]
Object Items	[massed Soviet armored divisions in the Fulda Gap]

In this simple example, the syntactic structure has 3 categories: actor, action, object. In more complex applications, the number of categories could increase.

Because different sentences can have the same meaning (e.g., "F-16s attacked tanks" means the same as "Tanks were attacked by F-16s"), NL parsers determine the "deep" or underlying "meaning" of a sentence

* Of course, the example we use here is a simple one. The actual operation of an NL parser is more complex.

whatever its "surface" structure is. The challenge for NL parsers is to apply the right set of "grammar rules" so that sentences that look different but mean the same have the same deep syntactic structure.

In this section, we describe how we will modify and use an automated NLP developed by a member of our project team.

2.1 PARSE - A Thematic Encoder

We will use PARSE as our natural language parser.

PARSE was originally designed to analyze text dealing with actions between national actors. PARSE will be described in its original application to show how a natural language processor works. We will modify PARSE for the specific application to air readiness assessment.

PARSE differs from other parsing techniques in several major respects. (1) PARSE defines a "theme" rather than a sentence as the "end-state" for its analysis of input text. PARSE has the capability to recognize and process correctly several themes embedded in a sentence too complex for complete parsing. (2) The syntactic nodes of PARSE are not all the traditional linguistic constructs (e.g., "noun phrase" or "verb phrase"); some are unique constructions appropriate for the type of analysis anticipated (e.g., "national actor" phrase). Similarly, PARSE uses a set of grammar codes different from standard grammar codes. The unique grammar codes reflect the particular purposes for which the parser was designed. (3) PARSE uses a specially designed grammar whose rules are of three types: (a) standard transformation functions; (b) unique disambiguation functions to resolve complex grammatical elements into simpler, basic grammar units; and (c) pattern matching elements (e.g., determining whether "trade" is a noun, verb, or adjective) at an early stage of processing (i.e., before a deep structure is produced), thus increasing the efficiency and capability of the program.

(4) PARSE has been designed to use semantic information (e.g., certain characteristics of national actors, their attributes, or relationships with other national actors) by adding rules to the existing grammar without disrupting the normal parsing process. (5) PARSE has been designed for ease in modifying its grammar to take into account increases in the complexity of input themes; this design allows PARSE to "grow" easily as its file of parsed themes increases.

PARSE produces as its end-state a construct we have called a "theme." We conceive of a theme as some set of English expressions representing interactions between nations or entities of interest. The following expressions are examples of themes:

(1) Syria attacked Israel.

(2) ...Syria began to attack Israel...

(3) ...the attack on Israel by Syria...

(4) ...Syria attacking Israel...

(5) ...Syria by attacking Israel...*

(6) ...Syria to attack Israel...*

(7) ...Syrian attack on Israel...

Only (1) above is an actual English sentence. The remaining expressions are sentence fragments, nominalizations, infinitive structures, etc. For our purpose, though, each of the above represents an acceptable end-state for PARSE. In fact, all these themes may be interpreted, using PARSE codings, as representing the same relationships between

* Expressions (5) and (6) may be themes depending on the context in which they occur. If the context of expression (5) were to read "Syria, by attacking Israel, regained territory," (5) would be considered a theme. If the context were "Egypt surprised Syria by attacking Israel," (5) would not be considered a theme. Expression (6) is also dependent on context for inclusion as a theme. PARSE provides the necessary information to determine when expressions like (5) and (6) would be defined as themes.

Syria and Israel, i.e., Syria is an actor involved in an attacking action directed toward a target, Israel. Just as some parsers have defined their end-state of "sentence" as consisting of a noun-phrase and verb-phrase (with optional prepositional phrases), we have operationally defined "theme" as one of a limited set of patterns of grammatical constructs appropriate for representing interactions between nations.

As part of PARSE, a new grammar was developed, tailored to the task of identifying actor/action/target relationships among text elements, that permits complex grammatical structures to be identified easier than using transition-network grammars. Central to this new grammar is a model of analysis involving the parallel application to all suitable text element strings of an ordered sequence of transformation rules and structural patterns. In the transition-network grammars, the text elements are processed in a fixed order, one at a time from left to right; the sequence of text elements determines the order in which successful parsing rules are executed. When a rule fails, backup must occur far enough on the text string to permit an alternative rule to be applied. In PARSE, the grammatical rules are executed in a fixed order; the sequence of rules determines which text elements are processed. When a rule fails, backup is limited to just that one rule. Because the text is processed at whatever element is appropriate for the transformation rule being executed, the need for recursion and backup is virtually eliminated.

To facilitate development of the parser and to simplify its revision and extension, PARSE was designed as an interpreter that invokes a grammar supplied to it in a separate file. A special language was designed to represent the grammar. This language has only a small number of operators, greatly simplifying the task of writing the grammar. The grammar references a set of grammatical labels. Words in a master encoding/recoding list are linked to a PARSE grammar dictionary in which each word is assigned a single grammatical label. This label may represent one or more grammatical codes. In some instances, these codes correspond to conventional grammar codes (e.g., "adjective," "adverb," etc.); non-traditional codes are used where special distinctions are required by the PARSE grammar. Simple grammatical labels represent just one grammatical code and may take arguments that further characterize the word (e.g., the word "on" has the label "PREP" with the argument "ON"). Complex grammatical labels are assigned to words that may take one of two simple grammatical labels (e.g., the word "call" would be a "complex" representing either a verb or a noun). Words with complex labels have two sets of arguments characterizing them; one for each of the possible simple labels. Generic grammatical labels represent classes of simple or complex labels; they are never assigned to words in the grammar dictionary, but are used to reduce the number of rules in the grammar. For example, some rules have the label "VERB" that refers to verbs of all types (e.g., transitive, intransitive, etc.). Although no word in the grammar dictionary is labeled "VERB," every verb grammar code is a member of that class.

The operations in the grammar can be summarized as follows:

o Find and process a text element having a designated grammatical label.

o Find and process a text element having a designated grammatical label and save its characteristics for later reference.

o Find and process a text element having a designated grammatical label and test its characteristics, either against previously saved characteristics from another element or against specified characteristics.

o Concatenate (join together) a preceding series of text elements and assign to the result a designated grammatical label, optionally with characteristics saved from one or more previous elements.

- o Define the thematic role of a text element just scanned successfully.

- o Iterate a designated operation any number of times.

Several additional operators are available that simplify the notation (such as an alternation operator that permits a set of alternative operators to be specified) but do not add any substantially new capabilities. The grammar, written in this notation, is processed by a special compiler that checks it for consistency with the syntax of the grammar-specifying language and produces a formatted ASCII file containing table entries that ultimately control the operation of PARSE.

One of the more important products of the grammar compiler is a set of tables that supports an optimizing function in PARSE. To avoid the combinatorial pitfall of applying every rule of the grammar to every element of the text, PARSE performs an optimizing operation that links each grammar rule only to those text elements that might satisfy the first test in that rule. This optimization is performed for every element of text in a sentence prior to applying the first grammar rule; thereafter, optimization is performed each time a concatenated sequence of text elements is assigned a new grammatical label. Although it might seem that in this process one set of combinations (rules by elements) in the application of all rules has merely been replaced with an equivalent set of combinations (elements by rules) in the optimization pass, a dramatic savings is actually effected. Because the decision as to which grammatical labels are to be linked to which rules is independent of the input text for any given grammar, it needs to be performed only once at compile time rather than for every sentence at execution time. The compiler produces a set of tables representing a linked-list structure that indicates, for each grammatical label, which rule(s) test for that label as the first test. As a result, the optimization operation can be keyed directly from the grammatical label of each text element and does not require any combinatorial operations. Admittedly, because optimization is based only on the first test in each rule, the procedure outlined above can be improved, but as a first cut it appears to strike a good balance between complexity and precision. If subsequent analysis of the performance of PARSE indicates it would be advantageous, the procedure can be extended to utilize pairs of elements, triples of elements, or a sequence of elements as long as is required by each rule.

The first section of PARSE reads the file that contains the tables that drive the PARSE interpreter. If a file containing a binary representation of these tables is available, it is read into the driving tables directly. If the binary file is not found, a file containing an ASCII representation is processed, producing a set of driving tables in binary that is then written out as a binary file that can be read in with no conversion required the next time PARSE operates. A new grammar can be invoked simply by compiling that grammar (thus producing an ASCII representation of the PARSE driving tables) and deleting any existing binary file of the driving tables.

The second section retrieves the numerically encoded elements of a sentence to be analyzed. The code value for each element points into a recode list that in turn is linked to a dictionary containing the grammatical label assigned to that element together with an additional set of characteristics. This information is retrieved and stored in tables linked to the element. The optimizer is called to link rules to the element, where appropriate.

When the properties of all elements in a sentence have been acquired, the third section is entered, where the rules of the grammar are processed in the prescribed order. Each rule is applied to all elements to which it has been linked by the optimizer. If the rule fails for any reason, any changes it produced are "undone." Successful rules cause one or more elements to be assigned a new grammatical label (with a consequent call on the optimizer) and/or cause one or more elements to have a theme role defined. When the last rule has been applied to the last element linked to it, the sentence has been analyzed as thoroughly as possible.

After applying all appropriate grammar rules, the fourth section is entered, where the results of the analysis of the sentence are displayed to the user and added to an output print file. In addition, codes representing the role(s) of each element in each theme encountered are added to a file. Sections two through four are repeated until all sentences from the inversion list have been processed.

PARSE GRAMMAR CODES. The grammar code dictionary has been gradually expanded to its present size of more than 18,000 unique entries. Every entry has a grammar code associated with it that is appropriate for processing by the PARSE grammar discussed below.

PARSE GRAMMAR. The PARSE grammar consists of a sequence of rules, each of which contains a sequence of operations. There are two distinct sets of rules: one set (consisting of disambiguation and transformation functions) identifies the components of a sentence that constitute a theme; the second set (consisting of pattern-matching and transformation functions) defines themes in terms of particular combinations of components.

Theme Component Identification. The first set of rules (now consisting of 84 rules) identifies and isolates the components of a theme. Within this set, there are two types of rules: (1) "transformation rules" which may assign a new grammatical label to two or more contiguous elements and (2) "disambiguation rules" which resolve a complex verb/noun ("CPXVN") into its respective verb or noun type. Operating on the grammar label for each word in the sentence, the grammar rules are applied in a prescribed order only to those elements that may be either transformed or disambiguated. The application of the complete set of rules results in the identification of the following elements that can be parts of a theme:

(1) NAC. An actor phrase, composed of a weapon system, munition, mission or national actor and a set of associated attributes or modifiers.

(2) Verb elements. Verbal elements consisting of all "transitive" and "intransitive" type verbs with their modifiers (e.g., TRNSV, ITRVFC, etc.).

(3) INFIN. Infinitive phrases (e.g., "...to attack...," etc.).

(4) NOMLZN. A nominalization, consisting of an "emphatic" noun and its prepositional complement, which represents an "action" link between two actors (e.g., "...denunciation of...," "incursion into...," etc.) and associated modifiers.

(5) Prepositional Phrases. Prepositional phrases are of three different types:

(a) Prepositional phrases that do not contain an actor or nominalization (designated PREPPHR).

(b) Proper noun prepositional phrases (PNPREPPHR) that do contain an actor.

(c) Prepositional phrases that contain a nominalization (PREPNOMLZN).

Theme Definition. The second set of rules (now consisting of 16 rules) recognizes particular patterns of the previously identified elements as theme components and defines their role within the theme as one of the following:

(1) Actor (indicated by a /3*)
(2) Actor attribute (indicated by a /3)
(3) Auxiliary action (indicated by a /4A)
(4) Action (indicated by a /4)
(5) Target (indicated by a /7*)
(6) Target attribute (indicated by a /7)
(7) Direct object (indicated by a /5*)
(8) Direct object attribute (indicated by a /5)
(9) Indirect object (indicated by a /6*)
(10) Indirect object attribute (indicated by a /6)
(11) Locational reference (indicated by a /8*)
(12) Locational reference attribute (indicated by a /8)
(13) Issue area reference (indicated by a /9*)
(14) Issue area reference attribute (indicated by a /9)
(15) Area/Issue reference (indicated by a /1*)
(16) Area/Issue reference attribute (indicated by a /1)
(17) Prepositional phrase element (indicated by a /P)

While other parsers define their end-state of "sentence" in relatively simple terms (i.e., a "noun phrase" followed by a "verb phrase"), the processes by which these terms are identified constitute almost the entire parsing efforts. PARSE, on the other hand, defines its end-state or "theme" in somewhat more complex terms (e.g., a NAC followed by some specified combination of verb elements or INFIN or NOMLZN followed by a NAC or some type of prepositional phrase), but greatly simplifies the process of identifying these thematic components.

In the next subsection, we show how PARSE works by going through each step in processing a sentence.

OPERATION OF PARSE. The following sentence and its grammatical codes are represented in Figure 3-1 below:

(1) UAR warned if Israel proceeds with atomic weapons Arab states must launch preventive war.

WORD	DICTIONARY LABEL (Grammar Code)
UAR	PROPN(UAR)
warned	TRNSV
if	PAUSE
Israel	PROPN (ISR)
proceeds	CPXFC (with, against;-)
with	PREP (with)
atomic	ADJ
weapons	COMN
Arab	PROPN(ARAB)
states	CPXNFC(to;-)
must	BE
launch	CPXOC (against, toward, on;-)
preventive	ADJ
war	EMPHN (on, with, against)

Figure 3-1

Operating on the grammatical codes for each word of the input sentence, PARSE applies the grammar rules in a prespecified and unvarying order to all appropriate elements within the sentence. These are indicated on Figure 3-2 as I1-I9 (for Identification Rules) and D1-D3 (for Definition Rules).

The first set of rules is designed to disambiguate complex elements (words which may be either a verb or noun) and collect and identify the components of a theme. The first successful application of a rule to the sample sentence disambiguates the word "proceeds" by scanning the next grammatical code to the right of "proceeds" and finding a preposition that matches the prepositional complement ("with") of the verbal element in the complex form to its verbal form: it now becomes an intransitive verb that takes a "to" complement with an optional prepositional complement of "with."

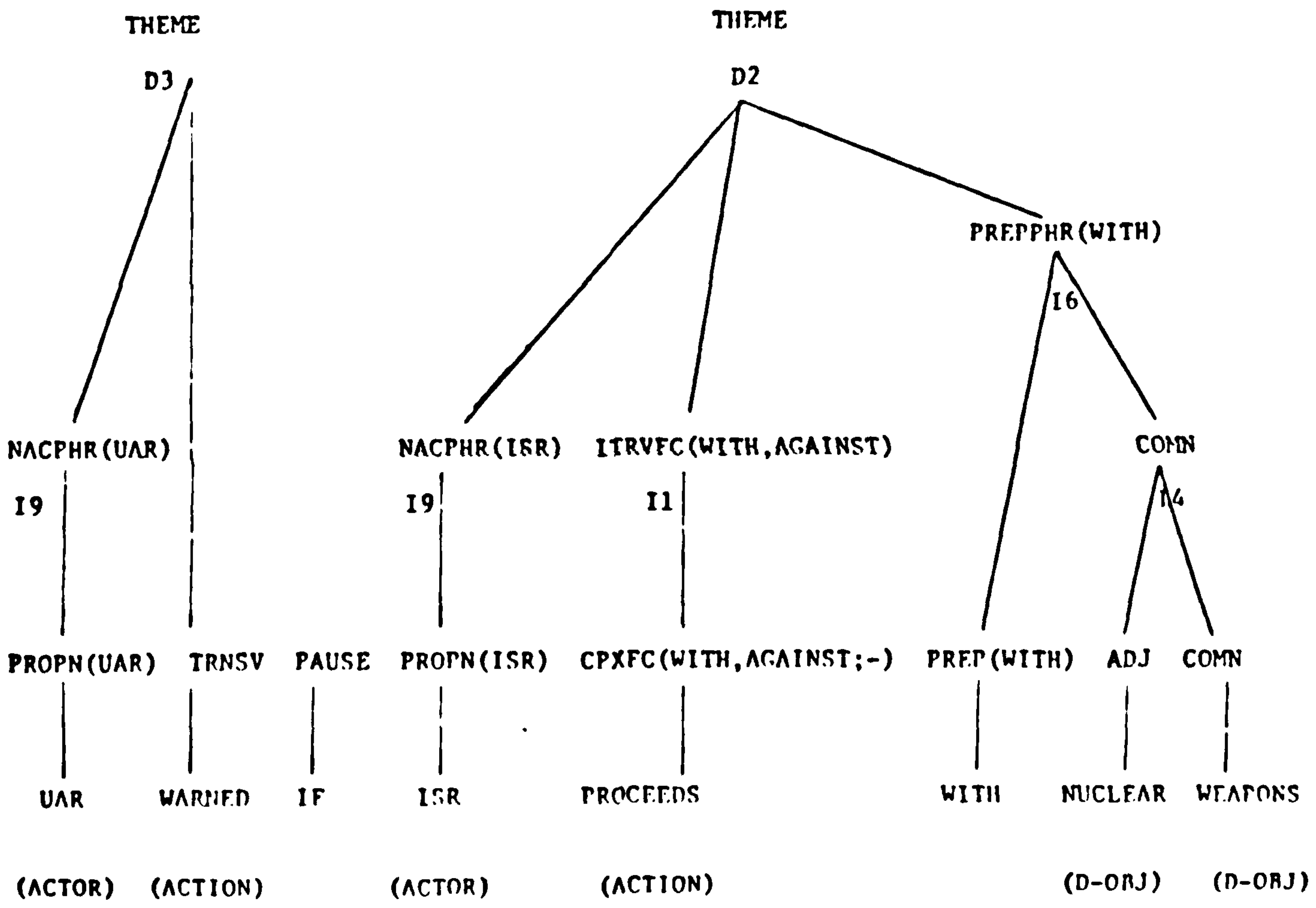
The second successful application of a rule disambiguates the word "states" by scanning the next grammatical code to the right and finding the auxiliary verb, "must." Any auxiliary verb immediately following a complex resolves that complex into a common noun.

The third successful application of a rule determines that the word "war" is not part of a nominalization. This is done by scanning the next grammatical code to the right and finding it not to be an appropriate prepositional complement for an emphatic noun (i.e., "on," "with", or "against"). This failure to find an appropriate complement changes the emphatic noun to a common noun.

The fourth successful application of a rule starts at the word "atomic." Locating an adjective and scanning to the right to find the common noun "weapons," both elements are concatenated into a common noun. The same rule also applies successfully to the adjective "preventive" and the common noun "war," concatenating them into a common noun.

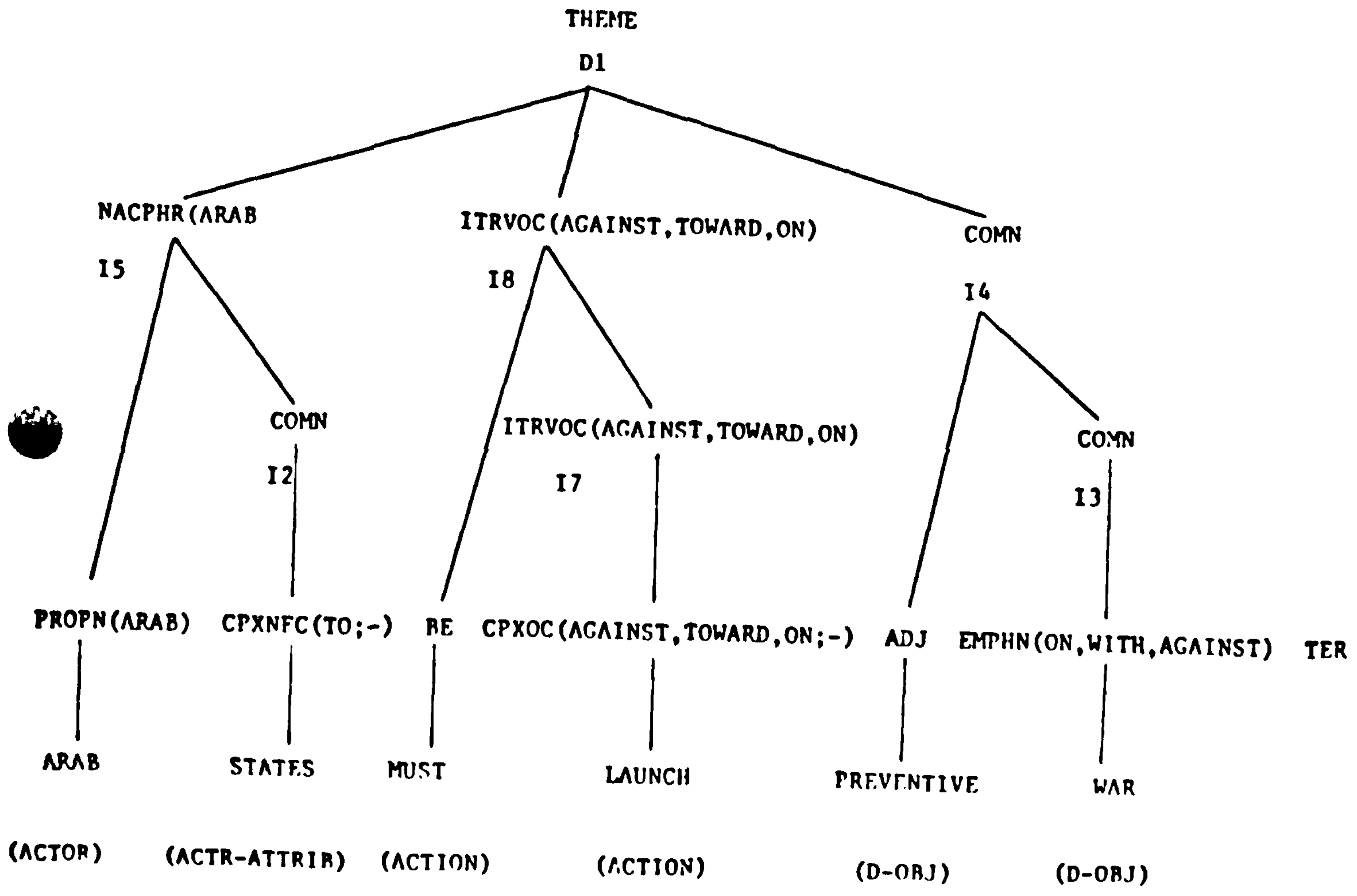
The fifth successful application of a rule combines the words "Arab" and "states." When a proper noun is immediately followed by a common noun, they are concatenated into an actor phrase (NACPHR) containing the argument of the proper noun ("Arab").

The sixth successful application of a rule is the preposition "with" and the words dominated by the immediately following common noun code (i.e., "atomic" and "weapons".) A preposition followed by a common noun (or common noun node) are both concatenated into a prepositional phrase (PREPHR) containing the argument of the preposition ("with").



Parsed Structure of Sentence

Figure 3-2



Parsed Structure of Sentence

Figure 3-2

(Continued)

The seventh successful application of a rule resolves complexes following forms of the verb "be." A grammatical label "BE" is found at the word "must." A complex following a "BE" is disambiguated into its verbal form: "launch" now becomes an intransitive verb with optional, prepositional complements of "against", "toward", and "on" ["ITRVOC (against, toward, on)"].

The eighth successful application of a rule combines the words "must" and "launch." A "BE" followed by a verb are both concatenated into a verb of the same type.

The last successful application of the first set of rules* is to the two remaining proper nouns. Each is redefined as an actor phrase having as their respective arguments the arguments of the original nouns.

At this point, the first set of rules has collected and identified the constituent elements of our thematic units: (1) actor phrases (NACPHR); (2) verbal elements (in this case, ITRVFC and ITRVOC); (3) common noun phrases (COMN); and (4) prepositional phrases (PREPHR). A second set of grammar rules is designed to recognize the appropriate pattern of thematic constituent elements and assign to each element its thematic role: actor, action, target, etc.

The first theme identified is that of an actor phrase followed by a verbal element followed by a common noun (i.e., the theme "Arab states must launch preventive war"). Every word in the sentence dominated by each of these nodes is given a different role designation: (1) NACPHR words that were originally labeled PROPN are identified as ACTORS (i.e., "Arab"), while other words subsumed by that node are identified as ACTOR-ATTRIBUTES (i.e., "states"); and (2) verbal element words are identified as ACTIONS (i.e., "must" and "launch"); and (3) common noun words were identified as TARGETS (i.e., "preventive" and "war").

* There are, of course, unsuccessful applications of rules. For example, the rule that successfully disambiguated the code for "proceeds" was applied unsuccessfully to the code for "states".

The second theme identified is that of an actor phrase (i.e., "Israel") followed by a verbal element (i.e., "proceeds") and a prepositional phrase (i.e., "with atomic weapons") of a type that matches the prepositional complement of the verbal element (in this case, "with"). These three elements of the theme are respectively identified as ACTOR, ACTION, and TARGET.

The third theme identified is that of an actor phrase (i.e., "UAR") followed by a transitive verb (i.e., "warned"). These two elements are identified as ACTOR and ACTION respectively.

In addition to identifying the individual themes in a sentence, PARSE maintains the coherence of the entire sentence so that the separate themes are related. Thus, the first two themes identified (i.e., the phrase "...if Israel proceeds with atomic weapons Arab states must launch preventive war") can be seen to be the object of the "warning" in the last theme identified ("UAR warns"). The final output is shown in Figure 3-3.

```
INPUT SENTENCE:  UAR warned if Israel proceeds with nuclear weapons Arab
states must launch preventive war.

CODED SENTENCE:  UAR/++3* warned/++4 if Israel/+3* proceeds/+4+ with/+7+
atomic/+7+ weapons/+7+ Arab/3++* states/3++ must/4++ launch/4++
preventive/7++ war/7++

THEME 1:  Arab/3* states/3 must/4 launch/4 preventive/7 war/7

THEME 2:  Isr/3* proceeds/4 with/7 nuclear/7 weapons/7

THEME 3:  UAR/3* warned/4
```

Figure 3-3. Coded Output Sentence in Figure 3-1

Two important aspects of the PARSE operation should be noted. First, since all complex codes were resolved before that element was concatenated with any other element, at no time was a tree structure built and then rebuilt in searching for an acceptable end-state. Second, prepositions played an important role in differentiating between components of a theme that could alternatively be labeled either as a simple prepositional phrase or as a prepositional phrase that is the object of some action. This second point should be clear if we consider a slightly different sentence from the one above:

(2) UAR warned if Israel proceeds along this path Arab states must launch preventive war.

The prepositional phrase, "...along this path...," would be labeled as a prepositional phrase associated with the theme, "Israel proceeds...," but it would not be labeled as a direct object of the action because its preposition does not match the required prepositional complement of "proceeds" (i.e., "with"). This linking of prepositional complements with the prepositions in prepositional phrases to determine the relationship of the phrase to the theme is an important aspect of the PARSE operation.

3.2 The Semantic Component of NLP

A major development in applied natural language processing (NLP) is the use of "case frame instantiation." This computational method utilizes knowledge of both the structure and the meaning of the sentence being processed, just as humans do. In addition, it facilitates handling of sentences that are either incomplete or ungrammatical. This approach, and its advantages over other approaches, will be discussed and illustrated here.

Case frames consist of a head concept, roughly equivalent to the main verb of the sentence, and a set of roles, or relationships to the head concept. For example, in the following sentence

"George broke the window with a hammer for Mary in Chicago"

the head concept is "break", and the other roles present are: _agent_ "George", _object_ "window", _instrument_ "hammer", _benefactee_ "Mary", and _location_ "Chicago". The following case frame has been "instantiated" or filled in, based on the previous sentence:

```
[BREAK                              (CF 1)
    [cases
        AGENT:  George
        OBJECT:  window
        INSTRUMENT:  hammer
        RECIPIENT:
        DIRECTION:
        LOCATION:  Chicago
        BENEFACTEE:  Mary
        CO-AGENT:     ]
    [modals
        TIME:  past
        VOICE:  active]]
```

The above case frame would also be instantiated when the sentence

"The window broke"

is processed. Note that "window" still fills the object case, whereas the agent case is unfilled. Thus, natural language processing using case frames requires knowledge about the meaning of the words, as well as about their position in the sentence. Although in the first sentence "window" is the syntactic direct object, while in the second sentence it is the subject, it fills the same semantic case in both sentences.

Each case frame defines some cases which are required, some which are optional, and some which may be forbidden. For example, the agent case in the example "The window broke" is unfilled, yet this is a complete sentence, demonstrating that the agent case for "break" is optional. On the other hand, "break" requires an object case, without it a sentence is incomplete.

The questions which a user might ask of an expert system to assess tactical air capability can be effectively processed using the method of case frame instantiation. A case frame, which might be constructed to account for a question type frequently encountered by our system is outlined below.

```
[PRODUCE                                   (CF 2)
    [cases
        AIRBASE (optional)
        SORTIE (required)
            (quantity, mission-type, aircraft-type...)
        GOAL (optional)
        DURATION (optional)

    [modal
        TIME:
        VOICE:
        MOOD:  (e.g. imperative, "wh" question, "yes-no" question)]]
```

Such a case frame may be used to account for a variety of sentences, e.g.

1. How many OCA sorties can Hahn produce in the next 7 days?

2. Give me the number of sorties that Bitburg can generate tomorrow?

3. How many F-16 sorties can I get from the Central Region in the first 30 days?

4. Can I produce enough sorties to meet the tasking?

In addition to a "produce" frame, instantiated by questions concerning number of sorties produced, other case frames can be designed to handle questions about sorties, e.g. increasing sortie production. In the above example, instead of more general cases such as agent, object, etc., the actual names of typical roles, AIRBASE, SORTIE, GOAL, and DURATION have been used. Furthermore, as illustrated in the examples, both the cases and the head concept are used to stand for any word or phrase which might express them. For example, the case frame instantiation would be set in process if the wording were "produce sorties," "get sorties," or "generate sorties."

Other case frames which might be designed specifically for the tactical air capability expert system involve questions about the location, existence, amount, sufficiency, incrementation, or depletion of a resource, e.g. aircraft, POL, munitions. For example, the following case frame concerns resource quantity.

```
[EXIST
    [cases
        RESOURCE (required)
            (quantity)
        AIRBASE (optional)
        GOAL (optional)
    [modals
        TIME:
        VOICE:
        MOOD:]]
```

This case frame is intended to account for questions such as

1. Where are my F-16s?

2. Is there sufficient runway to launch and recover the aircraft?

3. Are there enough dumb bombs to support 140 sorties tomorrow?

4. Are there any equipment shortages at Hahn?

5. How many AGM-45 missiles·are there at Spang?

In addition to the header and the collection of semantically defined cases, an implemented case frame also includes information about the actual wording of the sentence being processed, and about the position in the sentence of each case. During case frame instantiation the positional information is used to locate the filler of a case.

When the user types in a question, e.g., "How many AGM-45 missiles are there at Hahn?, the following processes occur:

1. Each case frame in the grammar is examined to find a match between the header and the words in the question. If no match is found, the question is not understood. If there is a match (or matches), the next processes take place.

2. *The case* frame (or frames) whose header is matched is retrieved.

3. An attempt to recognize each required case is made. If any required cases are not recognized, alternative processing is invoked; the user's question may be incomplete or ungrammatical, and must be processed further before the case frame can be instantiated.

4. After the required cases have been recognized, an attempt to recognize the optional ones is made. Failure to recognize optional cases does not result in the invocation of alternative processing.

5. If, after recognizing required and optional cases, any other words remain unrecognized in the question, alternative processing is invoked to determine why.

For example, if the user asked the following questions:

"How many sorties can Spang generate in a week?"

"And Hahn generate?"

the first question would be successfully processed and a case frame resembling CF 2 above would be filled, i.e.

```
[PRODUCE
    [cases
        AIRBASE    Spang
        SORTIE     sorties
          (quantity requested, mission-type __, aircraft-type __)
        GOAL
        DURATION   a week
    [modals
        TIME:    present
        VOICE:   active
        MOOD:    "wh" question count]]
```

However, the second question would not be successfully processed initially. Instead, alternative processing would be invoked; it would be determined which if any case frames were partially instantiated, and the previous dialogue would be examined in order to complete the case frame instantiation if possible.

The advantages of case frame instantiation over other methods include (1) the combining of bottom-up recognition of individual words and phrases with top-down recognition of the semantic components of a sentence. This facilitates more efficient and broad understanding of incomplete and ungrammatical sentences; (2) case frames combine knowledge of structure and knowledge of meaning, thereby reducing structural and lexical ambiguity; (3) case frames can also be used to generate natural language output to the user, in contrast to other methods which are suitable only for recognition/understanding of input, or for generation of output, but not both.

IV. Knowledge Base

It is axiomatic that the power of an expert system is proportional to the quality and scope of its knowledge.

Information in this knowledge base takes several different internal forms:

- o semantic network defining relationships between objects;
- o rules for accessing and manipulating the semantic network;
- o procedures for conventional processing; and
- o Frames for defining objects. This is inseparable from the graph-like semantic network.

The rules and the semantic network, viewed as a single Knowledge Base (KB), can be partitioned into domain-specific and domain-independent knowledge. Domain-specific knowledge allows the system to understand the application. The latter allows understanding of its own approach. If the system can be powerful enough to know its own scope limitations, then it will know when a question falls outside its area of expertise. As discussed below, this integration of types of knowledge will help the natural language front-end.

4.1 Semantic network

The semantic network represents every known object in the defined "world," in a nearly one-to-one correspondence. Objects are defined by their attributes and relationships to other objects. Attributes are important characteristics of objects, like size and color. The relationships between objects tie the KB together. Semantic networks are potentially unbounded in size; there are always more attributes that can be added, more relationships to note, additional objects to define. Domain-specific knowledge about the world of TAC assessment resides with knowledge about the semantic network itself.

To provide robustness in the natural language front-end, the knowledge base must possess general familiarity with a large body of AF knowledge. The semantic network will contain information on:

TAC organizations
TAC data bases
weapons systems
air bases
missions
munitions
supplies

Since the system will understand the limitations of its knowledge, it can respond intelligently to questions that it knows it cannot answer. The information in the KB is roughly bounded by the scope of the effort (that is, the four air bases, the aircraft at those bases, the missions typically flow by those aircraft, etc.). The objects in the TAC-specific portion of the KB include information mentioned by the domain expert or informed by the system developers. Those objects, attributes, and relationships mentioned by the expert are the information most important to the expert's problem solving strategies. Other objects and relationships added by the knowledge engineers can be classified as "obvious" or "intuitive." This latter information will not normally be referenced by the expert's rules, but will provide the semantic links between general questions and specific information.

Other objects and relationships outside the problem domain are better classified as "meta-knowledge" ("knowledge about knowledge"). Meta-knowledge consists of very general, high-level objects that loosely tie the network together, sometimes also called "abstract concepts." An effective interface between the NL parser and the expert system should "understand" what an object's attributes mean and how to use the relationships that link objects. The system will link the components of the question and the relationships of objects in the network. For example, suppose that the object "F-4E" has an attribute named "MISSION-TYPES." To answer a trivial question such as "What can F-4's fly?" the system will have to establish a semantic link between "can ... fly" and the "MISSION-TYPES" attribute of the F-4E object. Since the same question could be asked any number of ways, the system must really understand the meaning of "MISSION-TYPES" so that it will be referenced at the appropriate times. Different approaches to solving this problem are discussed as design issues in the Design Methodology section on Knowledge Base Design.

Conceptually, the vocabulary of the NL parser is also part of the KB. The parser requires semantic information to develop a "deep structure" representation of a user's question that is independent of the way it was phrased. The semantic network links the literal word and the object or relationship in the semantic network. For efficiency, the vocabulary will be stored in a data structure designed for rapid access. Reference the next section for discussion of design issues for the vocabulary and semantic network.

4.2 Rules

Rules tell the expert system how to view and manipulate its knowledge. The general form is:

IF (conditions) THEN (conclusions) DO (actions)

In expert systems, rules can be used in a "forward" direction from facts to conclusions or a "backward" direction. In the forward direction, new facts cause automatic actions. This means that if the specified (condiitions) can be shown to be valid, the (conclusions) are asserted in the knowledge base (knowledge is added, deleted, or modified), and optionally some (actions) are performed that affect the world outside the knowledge base.

This may in turn cause other rules to fire like a cascade. Forward rules thus propagate the effects of new information throughout the data base. (This propagation is also known as "chaining)." As an example (not actual), consider the following forward chaining rules:

```
1. If DEFCON-STATUS is DEFCON-3,
   Then
       ALERT-PERCENTAGE is 75% and
       PERSONNEL-ACTION is RECALL

2. If ALERT-PERCENTAGE is greater than 50% and less than 100%
   Then
       WARGOERS are ACTIVE and
       ALERT-PERCENTAGE is 100%

3. If PERSONNEL-ACTION is RECALL or
       MAINTENANCE-STATUS is CRITICAL
   Then
       SHIFT-LENGTH is DOUBLE
```

Rule 1 captures the knowledge that DEFCON-3 requires 75 percent of available aircraft to be on alert, as well as a personnel recall. If that rule fires because of a change in DEFCON status, then those two facts will be added to the KB. The first new fact causes Rule 2 to fire, which represents the "rule of thumb" that if more than half of the aircraft must be on alert, then go ahead and place all of them on alert, and start considering "wargoers" as available for employment. Rule 1 also causes rule 3 to fire, resulting in a double work shift.

Backward-chaining can proceed from known results to hypothesized causative agents. They have utility in question-answering situations. Desired information may not exist as a known fact in the KB, but instead may be a "conditional fact" which depends on the validity of a number of other facts. Referring to the general rule form above, if we have a desired fact and can find a rule with that fact as part of the rule's (conclusions), then we may consider the fact valid if we can show that each (condition) in that rule is valid. Each (condition) may be a known fact in the KB or a "conditional fact" that itself must be proved by finding another rule whose conclusions contain this other "conditional fact." This "backward chaining" of rules results in a search through the KB that traces a chain of evidence leading to the answer to the original query. As another example, consider the same three rules as above, but with the following query:

Are double shifts necessary in the current situation?

The system first looks for a fact that "SHIFT-LENGTH is DOUBLE." Failing that, it sees if it is derivable from the facts and rules in the KB. The translation from the question to the internal form (SHIFT-LENGTH is DOUBLE) will be handled by the interface between the natural language processor and the expert system. For this example, Rule 3 matches the query, so the system attempts to show that "PERSONNEL-ACTION is RECALL" or "MAINTENANCE-STATUS is CRITICAL" can be derived from the KB. The first possibility (RECALL) results in the firing of Rule 1, which causes the system to see if "DEFCON-STATUS is DEFCON-3." If it is, the system will answer that double shifts are necessary. The backward-chaining inference mechanism also gives the system an ability to justify itself by explaining the chain of evidence supporting that answer.

To explain the above example, the system would inform the user that:

DEFCON-STATUS is DEFCON-3.
DEFCON-3 implies that PERSONNEL-ACTION is RECALL.
RECALL implies that SHIFT-LENGTH is DOUBLE.

As these two examples show, the same rules may be used in both the forward and backward diretions. Forward rules respond to changing information, while backward rules support question-answering.

The rules will include both expert domain knowledge and general problem-solving knowledge. The expert knowledge (rules of thumb) provides shortcuts to problem solutions, guiding the discovery process at a high level. In the TAC assessment application, the expert rules will establish an environment for answering questions about TAC readiness when the answer is not immediately available.

The rules that embody general problem-solving knowledge are also classified as "meta-knowledge." Questions cannot be answered directly by using the meta-knowledge in the semantic network. Instead, certain expert rules must be triggered to transform a question into an appropriate scenario (hypothetical or actual) and an equivalent query that the expert rules can deal with.

4.3 Procedures

Procedures will play a limited role in this system, since they require separate development and debugging. In the few instances when our domain expert solved a problem using an approach that is naturally expressed as a procedure, it is important that the expert's problem-solving knowledge be encoded as directly as possible. Examples of likely procedures include scheduling algorithms (sortie generation, aircrew assignment) and simulation or processes (aircraft turnaround). The procedures will link into the KB through the semantic network and higher level rules.

4.4 Design Methodology

The following shows the approach we will take in developing the system. When design issues remain, the following subsections will discuss alternative approaches and tradeoffs. The design of an expert system differs from traditional software development. Section 3.1 provides an overview of the design process in the form of a diagram. The remaining sections describe a preferred approach to building the natural language processor, the interface between the NL processor and the expert system, the inference engine, the knowledge base, and the output text generator.

4.5 Knowledge Base (KB) Design

Design of the KB requires specifying the roles of the semantic network and the rules. Some of the knowledge will be most easily represented in the semantic network, other parts will be best implemented as rules. Much of the knowledge could be stored in either form. The tradeoffs between representing knowledge as rules or objects in the semantic network are discussed in the section on design issues for the KB. In general, the KB must be easy to expand. The knowledge base design must be general enough to permit frequent massaging of the rules and semantic network.

Efficiency considerations compel the KB builder to consider partitioning information into KB subsets, so that the inference engine runs faster in a smaller "search space" and is divided into modules. Semantic networks are, by their nature, partitioned, since semantic links connect those objects most closely related. It is crucial that the design include a hierarchy of rule classes. The partitioning of rules is by context: only those rules that are most appropriate for the problem at hand should be considered by the inference engine. The context will generally change as a result of rule conclusions.

The overall design of the KB is based on "object-oriented" programming. Viewing all members of a KB as specialized objects provides many benefits in terms of program development and user interface. With this approach, rules will in fact be part of the semantic network, partitioned from the remainder of the KB because rules are a separate kind of object from the rest of the objects in the KB (aircraft, missions, organizations). However, because rules are objects in the semantic network, they can be created and tested using the same tools that create the rest of the network. The benefit is seen primarily in the development phase; the user interface will hide all implementation details.

4.6 The Semantic Network

Objects in the semantic network are defined by their attributes (characteristics) and the relationships with other objects in the network. Usually the two most important types of objects in the semantic network are classes and instances. A class is a general category of object, such as AIRCRAFT. Instances are members of classes, specialized from the more general category. An example would be F-4, an instance (specialization) of AIRCRAFT. However, it is also apparent that the F-4 object, while a specialization of AIRCRAFT, is itself a broad category; it can be further specialized into F-4G and F-4E. Thus, an object can be both a class and an instance. The primary benefit of this relationship (called the IS-A or A-KIND-OF or SUPER-CLASS relationship) is one of inheritance. Attributes of more general classes are efficiently inherited by the specialized classes, and specialized classes can add attributes when it becomes important to do so. For instance, AIRCRAFT objects might have attributes called ENGINE-TYPE and NUMBER-OF-ENGINES. The various instances of AIRCRAFT (F-4, F-16, F-15) would inherit those two attributes, although the values of the attributes would in many cases vary from instance to instance. The F-4E to F-4G objects would inherit the attributes and values for the ENGINE-TYPE and NUMBER-OF-ENGINES attributes directly from the F-4 object.

Many other relationships beside IS-A (such as PART-OF) will exist in the network. One key goal of this effort is to recognize the critical relationships between pieces of knowledge in the KB and to implement those relationships so that correct and efficient reasoning is possible. The design and implementation of the semantic network will require extensive discussions with the domain expert; usually these will be the same discussions used to extract the important rule-based knowledge from the expert. Out of these talks will come a large body of uncorrelated knowledge. There will be an ongoing effort to categorize, refine, and load the information into the KB. Categorizing the knowledge involves several steps, repeated continually:

1. Look for "objects" in the loose knowledge.

2. Fix the object's position in the KB by abstracting the object to assign a general class for the object.

3. Look for attributes and relationships to other objects in the KB. This may include probabilistic reasoning with uncertain facts, or expressing "fuzzy" relationships or "what-if" scenarios.

4. Try to generalize the attributes and relationships to fit existing information. This will lead to the definition of meta-knowledge about those attributes and relationships.

5. Generalization of objects, attributes, and relationships for the "original" object may result in a recursive network-building process for those more general objects. As the KB grows, however, the abstraction process will simply fix the position of a new object in the knowledge base, either by establishing it as an existing object, or as a new specialization of an existing object.

As the knowledge base grows, similarities between physical objects and abstract concepts will become apparent allowing definition of higher level objects and relationships between classes. Although this will not decrease the size of the knowledge base (the lower level relationships will remain), it will allow the use of more powerful rules that reason with the knowledge at a higher level. Reasoning at such levels may approximate analogical reasoning in humans, allowing the system to use rules that cut across wide classes of objects in the KB. These higher level rules take the form of expert knowledge within the application domain, the meta-rules within the meta-knowledge. Any similarities drawn across apparently dissimilar classes can, in the proper context, greatly reduce the amount of searching the system must do for an answer.

V. REPRESENTATION AND USE OF RULES

Rules are a part of the system's knowledge base. The system represents rules as frames. Rules are stated in the form:

```
(IF    ([condition1]
        [condition2]
        .
        .
        .
        [conditionN])
 THEN  ([conclusion1]
        .
        .
        .
        [conclusionM])
 DO    ([action1]
        .
        .
        .
        [actionL]))
```

Rules are invoked if the conditions are all true. Rules are grouped into related classes so that not all rules need be considered for invocation at the same time. Actions are Lisp forms that are evaluated when a rule is invoked. Conclusions are arbitrary text or TellAndAsk expressions that are asserted when a rule is invoked. Asserting a conclusion means making it true in the knowledge base. TellAndAsk is an expression for asserting and deductively retrieving facts from the knowledge base. Conditions are TellAndAsk expressions, Lisp forms, or arbitrary text that are evaluated to see if a rule should be invoked. The conditions, conclusions, and actions in a rule can contain variables. When they do, the rule is invoked for each set of variable values that makes the conditions true. Conditions and conclusions can be connected by the logical operators AND, OR, and NOT.

Rules are used to create, access, and update the system's knowledge base. For example, when the Air Staff Officer using the system asks a question that cannot be answered directly from the knowledge base, rules are used to determine the answer by a process called backward chaining. Backward chaining is a reasoning process that starts by transforming the question into a statement to be proven true or false and works backwards to see if what is currently known will support the desired statement. If the statement contains variables, the backward chainer tries to find values of the variables for which the statement is true. The process the backward chainer uses is to start with the statement as its initial goal. Then it searches the knowledge base for rules whose conclusion part matches the goal. The matching rules condition part becomes a new set of goals that replaces the initial goal. The first of these goals becomes the basis for a second search, until no more goals remain (success) or no more goals can be removed (failure). In case of failure, the system reports a negative answer to the original question. In the case of success, the system invokes the rules that were used, starting with the rule that removed the last goal first. This process generates the answer to the question.

The efficiency of the backward chaining process depends in part on the search strategy used. The system can employ any desired search strategy including the three most commonly used strategies:

depth first search
breadth first search
best first search

In depth first search, each discovery of a rule whose conclusion part matches the current goal suspends the search for matches of that goal (it may be restarted later if the current path fails) and starts a search for a match to the first of the rules conditions. In breadth first search, all rules whose conclusions part matches a goal are found before the search is terminated. In best first search, an evaluation criteria is applied each time a matching rule is found. The search is continued using the goal that gives the most desirable evaluation.

Often, questions put to the system require that it determine some goals in order to provide an answer. Typically, the system may have to determine how many aircraft to use on a mission before it can answer whether an overall objective can be achieved. These situations are said to require planning abilities. Backward chaining is not suitable for them because the goal is unknown to start with. Instead, forward chaining is used. Forward chaining starts with the known facts and tries to deduce the desired goal. The process the forward chainer uses is to match the conditions of rules in the knowledge base against the known facts. When a match occurs, the conclusions of the rule are added to the known facts by invoking the rule. The process continues until no more rules apply. Among the deduced facts, we should be able to find the number of aircraft to use on the desired mission or else we will be unable to solve the problem.

The efficiency of the forward chainer is affected in part by the method used to select the rule to invoke next when there are several rules in the knowledge base that could be invoked. The system can use any desired method, including four common methods:

Invoke the rule having the least complex condition part.
Invoke the rule having the most complex condition part.
Invoke the rule having the greatest weight from among those having the least complex condition part.
Invoke the rule having the greatest weight from among those having the most complex condition part.

To avoid expanding the effort needed to draw unnecessary conclusions, a rule selection method that recognizes when the desired goal has been determined and can measure how close we are to getting there is more useful than one of the general methods listed above. In addition, structuring rules into classes that are focused on determining certain types of goals is a useful technique for reducing the potential effort involved in foreward chaining.

Rules are very useful in answering questions about why the system gave the answer it did. "Why"-type questions can be answered by telling the Air Staff Officer one or more of the chain of rules that were invoked to answer the question. For example, if the Air Staff Officer asks, "Why did you use F16s instead of F15s?", the system can answer, "Rules 274 states that if air-air and air-ground capabilities are required in the same airplane, then use F16s."

VI. DESIGN ISSUES

The crucial design issue is encoding knowledge: rules versus objectives. In the course of debriefing the domain expert, several examples of knowledge have been noticed that could be implemented either using rules or objects. One example concerns aircraft and missions. Aircraft can fly certain types of missions, and inversely, missions require certain types of aircraft. This information could be encoded in several ways: rules, objects, or as a table of data. The latter example is not considered here, although that decision may be revisited if efficiency becomes a critical issue. For encoding the information for F-4's and F-16's, the appropriate rules are:

```
If activity is PLANNING-MISSION and MISSION-TYPE is BAI,
        then PLANE-TO-USE is F-4G, F-16.

If activity is PLANNING-MISSION and MISSION-TYPE is SEAD,
        then PLANE-TO-USE is F-4G.

If activity is PLANNING-MISSION and MISSION-TYPE is DCA,
        then PLANE-TO-USE is F-4G.

If activity is PLANNING-MISSION and PLANE-TYPE is F-4G,
        then CAN-FLY-MISSION is BAI, SEAD.

If activity is PLANNING-MISSION and PLANE-TYPE is F-4E,
        then CAN-FLY-MISSION is BAI, SEAD.

If activity is PLANNING-MISSION and PLANE-TYPE is F-16,
        then CAN-FLY-MISSION is BAI, SEAD.
```

Using objects, we can define attributes in the classes AIRCRAFT and MISSION named CAN-FLY-MISSION and PLANE-TO-USE, respectively. The objects F-4G, F-4E, and F-16 inherit the attributes from AIRCRAFT, while the objects BAI, SEAD, and DCA inherit attributes from MISSION. The equivalent information, then, is:

```
F-4G:
        IS-A:               F-4
        CAN-FLY-MISSION:    SEAD
```

```
F-4E:
          IS-A:                F-4
          CAN-FLY-MISSION:     SEAD, DLA

F-16:
          IS-A:                AIRCRAFT
          CAN-FLY-MISSION:     OCA, CDA, BAI

BAI:
          IS-A:                MISSION
          PLANE-TO-USE:        F-16

DCA:
          IS-A:                MISSION
          PLANE-TO-USE:        F-15, F-16, F-4E

SEAD:
          IS-A:                MISSION
          PLANE-TO-USE:        F-4G, F-4E
```

There are several tradeoffs here. Encoding the knowledge as rules enables the system to explain how it used the rules to solve a problem (all inference engines that work with rules have this explanation facility). On the other hand, inference engines must generally consider a lot of rules when working, and encoding data as rules threatens to cause a bottleneck. Extensive use of rule classes might solve the problem, except most objects in the semantic network are useful across a wide array of rule classes; it is likely that even if the above rules were placed in a separate rule class (one for the context of missions), that rule class would be active often enough to negate the benefit of partitioning.

The design of this sytem will lean toward encoding data as part of the semantic network; rules will be scrutinized to ensure that they provide problem-solving knowledge, not merely data. We will attempt to limit rules to expert knowledge and meta-knowledge.

As mentioned above, the refinement of the knowledge base will require reworking some of the knowledge. Abstraction refinement has already been mentioned; this is done to the semantic network locally. There may also be refinement between the rules and semantic network. For instance, if a path through the semantic network becomes extensively travelled, it may be prudent to establish a rule that allows the system to take a shortcut or to actually define a new link between the ends of the well-travelled path. A semantic link may implicitly exist between two objects. Establishing that semantic link may result in improved performance. The example above (AIRCRAFT and MISSION relationship) shows that the data in the relationships can be encoded as part of the objects. Since the rules are actually solving the problem, the KB must still contain a rule to specify when the information is useful. This is an example of the power of higher-level rules; an appropriate rule in this case would be:

```
If activity is PLANNING-MISSION and we know PLANE-TYPE,
     then use CAN-FLY of PLANE-TYPE.

If activity is PLANNING-MISSION and we know MISSION-TYPE,
     then use PLANE-TO-USE of MISSION-TYPE.
```

These two rules will work regardless of the number of PLANE-TYPE's and MISSION-TYPE's. Using the data-as-rules approach above, a new rule would have to be added for each new AIRCRAFT or MISSION object.

6.1 Anticipated Problems.

There are three major parts of this effort that are likely to cause problems. The first problem will be to understand the meaning of user statements and questions to such a degree that action by the expert system is straightforward. The interface between the natural language parser and the expert system is uncharted territory, but we anticipate a very close relationship between the vocabulary of the parser and the semantic network. Since there are many ways to parse a question, the semantic processor will have to establish an equivalence between those many variations and a "situation" understood by the KB. Establishing this equivalence will require cooperation between the parser and the semantic network, with the vocabulary providing detailed information on how portions of the user's statement or question link into the semantic network, and which expert rules can be triggered by the user's input.

The second and third problems are closely related. Both the size of the knowledge base and the efficiency of access will become issues in the development of this system. To provide the interface robustness that we are seeking, the system will require a large knowledge base, with information on as many objects as are necessary to understand the vast majority of user questions. The semantic links (relationships) between these objects will be especially important because establishing the correct links will enable the system to respond efficiently and correctly. Speed will not be a significant concern, since the purpose of the system is to demonstrate the technology. However, we must have confidence that this demonstrated technology will "scale up" to a prototype system where response time and correctness are equally important.

One specific issue regarding the KB size and access efficiency is the representation of information stored in external TAC data bases and reports. Much of an expert's rote knowledge is raw data, available in numerous periodic reports and data bases but memorized over the years. It is unclear to what extent that large body of data can be duplicated in the KB. Such duplication would certainly be more efficient, but would clutter up the KB with infrequently used information. The KB must know about the information from those external sources, either to access the data directly, or refer to the sources indirectly. One approach may be to implement a crude model of human memory, in which real data will be stored in the KB as they are requested. If the system does not know the actual value, it will at least know where to find it. This will give the system the capability to "forget" about actual data when space becomes an issue, while retaining the memory of how to retrieve the forgotten information. Even representing the information in data bases at a high level will require significant room in the KB, but since external data bases are common sources of information for the Air Staff, such representation is vital.

VII. Meta-knowledge Network V-1

In an expert system, the expert's domain knowledge provides the power necessary to solve problems. It is often useful to incorporate more general knowledge into the expert system. Two benefits of this more general knowledge, or "meta-knowledge," are more efficient problem solving and a simplified interface to a natural language front end. Both of these benefits are important for the expert system in the TAC readiness assessment domain. Meta-knowledge can improve the speed of expert problem solving by reasoning about the problem in general, and by deciding which of the expert heuristics are most appropriate to the immediate problem. It can assist in interfacing an expert system to a natural language processor by providing a consistent high-level interpretation of the problem domain, into which the user's questions are translated, and from which the expert system can answer the questions.

The approach described here is to represent the domain of TAC sortie generation and readiness assessment as a network of general concepts, entities, and activities connected by a minimal set of general relationships. The network represents the most important knowledge about the TAC domain by identifying the critical concepts and the interaction between those concepts. In providing a robust interface to the expert system, it is crucial that the system understand the limitations of its knowledge. The boundary of the meta-knowledge is represented by concepts that are defined at the highest level of generality, or by concepts that are minimally connected to the rest of the network (and thus, are on the fringes of the network). Concepts within the scope of the effort are described in more detail, with more connectivity between other concepts in the network. The meta-knowledge network has the following purposes:

1. Represent relationships and interactions between high-level concepts and activities within the domain.

2. Provide a technique for designing meta-knowledge in an expert system.

3. Construct a framework for creation and organization of the experts' rules-of-thumb.

4. Assist in the identification of missing concepts and relationships.

5. Provide an environment capable of discovering concepts not explicitly represented in the network.

Minimizing the number of relationships (links between concepts) in the network results in a simple representation of knowledge. However, in order to ensure that the network is capable of representing the concepts and interactions correctly, it is important to choose the correct primitive relationships. Also, it is necessary to identify properties that permit the combination of primitive relationships to form application-specific relationships. The desire to minimize the number of possible relationships between concepts led to the identification of three relationships. These three relationships are capable of representing all concepts in the TAC domain, and permit interesting combinations that can represent other relationships. In the three items below, X and Y refer to two concepts X and Y.

1. Production (P).

X [P] Y	X produces Y X results in Y X determines (completely) Y

2. Consumption (C).

X [C] Y	X consumes Y X requires Y

3. Affection (A).

X [A] Y	X affects Y X determines (in part) Y

Refinements on these relationships permit the representation of desirable/undesirable situations. The meaning of desirable or undesirable occurs in the expert's heuristics, which amplify the general knowledge. In the representation of the network below, a plus sign means desirable, while a minus sign means undesirable. A relationship without refinement ([P], [C], [A]) can mean "... temporarily" or simply a general production, consumption, or affectation. For example,

[A+]	has a good effect on
[A-]	has a bad effect on
[C-]	destroys something good
[C+]	destroys something bad
[P+]	produces something good
[P-]	produces something bad

These relationships usually connect concepts in the domain network; however, it is both possible and very useful to allow relationships to connect concepts to other relationships or one relationship to another. An example in the TAC domain is that the mission type (BAI, DCA, etc.) determines, in part, which aircraft will be flown in a sortie. That can be represented as:

```
MISSION-TYPE [A] (SORTIE-GENERATION [C] AIRCRAFT)
```

which reads: SORTIE-GENERATION requires an AIRCRAFT, and the MISSION-TYPE, in part, determines or affects the requirement. To continue the example, weather and target affect the loading of munitions for a sortie, so:

```
WEATHER [A] (SORTIE-GENERATION [C] MUNITIONS-LOADING)
 TARGET [A] (SORTIE-GENERATION [C] MUNITIONS-LOADING)
```

Sections 7.1 and 7.2 identify the concepts and relationships compiled to date.

There are a number of properties of the P/C/A network that allow powerful reasoning about the concepts and explicit relations in the network.

1. [P] and [C] implies [A]. The production and consumption relationships between two concepts imply some sort of effect of one concept on the other.

```
X [P] Y  implies that  X [A] Y
X [C] Y  implies that  Y [A] X
```

That is, if one thing produces a second thing, that first thing affects the existence of the second. Similarly, if one thing consumes a second thing, that consumable affects the existence/ability/well-being of the consumer. The consumer also directly affects the thing consumed, just by consuming it.

The refinements (+/-) on [P] and [C] also carry over the implied effect [A]:

```
X [P+] Y  implies that  X [A+] Y
X [C+] Y  implies that  X [A+] Y
X [P-] Y  implies that  X [A-] Y
X [C-] Y  implies that  X [A-] Y
```

2. P/C/A relations are transitive. If X consumes Y and Y consumes Z, then, to some extent, X consumes Z. The same is true for production and affectation.

3. Good and bad effects ([A+] and [A-]) propagate. If Y affects Z and X has a good/bad effect on Y, then X will, to some extent, have a good/bad effect on Z.

4. Good and bad effects cancel. If X has a good effect on Z and Y has a bad effect on Z, then, to some extent, X and Y can cancel each other's effects.

5. The requirements of a producer affect that produced. If X [P] Y, then all Z such that X [C] Z have some affect on Y.

6. Negation of a bad effect X involves item 4 above (finding a Y that produces a cancelling good effect), or starving X, or eliminating the source(s) of X. That is, for:

X [A-] Z (X has a negative effect on Z)

a. Find a Y such that Y [A+] Z

b. Consume all Y such that X [C] Y

c. Consume all Y such that Y [P] X

7.1 Concepts, Entities, Activities in the TAC Domain.

AIRBASE-STATUS

AIRCRAFT

AIRCRAFT-STATUS

AIRCREW

AIRLIFT

ALERT-STATUS

ARMY

BAD-WEATHER

BAI

CIVIL-ENGINEERING

COMMUNICATIONS

CREW-REST

DCA

DEFENSIVE-HOSTILITY

DUTY-DAY

ESCORT

FLYING-SCHEDULE

FUEL-STORE

FUEL-TRUCK

GROUND-TRANSPORT

HOSTILITY

MAINTENANCE

MAINTENANCE-CREW

MECHANICAL-BREAKDOWN

MAC

MILITARY-ENVIRONMENT

MISSION-TYPE

MUNITIONS-ASSEMBLY

MUNITIONS-ASSEMBLY-CREW

MUNITIONS-LOADING

MUNITIONS-LOADING-CREW

MUNITIONS-STORE

OCA

OFFENSIVE-HOSTILITY

PIPELINE

POLITICAL-ENVIRONMENT

RECCE

RECOVERY

REFUELING

REFUELING-CREW

RUNWAY

SCL

SEAD

SORTIE-GENERATION

SPARE-PARTS

SUPPORTING-AIRBASE

SUPPORT-FOR-TASKING

TACTICS

TASKING

THREAT

TIME-FRAME

TRAINING

7.2 P/C/A Relationships in the TAC Domain.

In the list below, the concepts listed in Section 7.1 are related to other concepts using the plain and refined P/C/A relationships described in Section 7. A blank left-hand side of the relationship implies the same left-hand side as the one above it (see the three MILITARY-ENVIRONMENT relationships in lines two through four, below). The relationships are grouped according to the figures they appear in.

Figure 1.

```
POLITICAL-ENVIRONMENT  <A>  MILITARY-ENVIRONMENT
 MILITARY-ENVIRONMENT  <A>  ALERT-STATUS
                       <A>  OFFENSIVE-HOSTILITY
                       <A>  DEFENSIVE-HOSTILITY
                       <A>  TASKING
  OFFENSIVE-HOSTILITY  <P-> THREAT
  DEFENSIVE-HOSTILITY  <P-> THREAT
               THREAT  <A>  TACTICS
              TACTICS  <A>  TASKING
```

Figure 2.

```
DEFENSIVE-HOSTILITY <C-> AIRCRAFT
OFFENSIVE-HOSTILITY <C-> AIRCRAFT
                    <C-> COMMUNICATIONS
                    <C-> FUEL-STORE
                    <C-> MUNITIONS-STORE
                    <C-> SPARE-PARTS
                    <C-> RUNWAY
                    <C-> PIPELINE
                    <C-> AIRLIFT
                    <C-> GROUND-TRANSPORT

(HOSTILITY <C> AIRCRAFT) <P>  AUGMENTATION
            AUGMENTATION <P+> AIRCRAFT

   DCA <C+> OFFENSIVE-HOSTILITY
  SEAD <C+> DEFENSIVE-HOSTILITY         ; SAM
ESCORT <C+> DEFENSIVE-HOSTILITY         ; ENEMY DCA
   BAI <C+> OFFENSIVE-HOSTILITY
       <C+> DEFENSIVE-HOSTILITY
   OCA <C+> OFFENSIVE-HOSTILITY
       <C+> DEFENSIVE-HOSTILITY
```

Figure 3.

```
   BAD-WEATHER <A-> AIRBASE-STATUS
               <A-> RUNWAY
COMMUNICATIONS <A>  AIRBASE-STATUS
    FUEL-STORE <A>  AIRBASE-STATUS
        RUNWAY <A>  AIRBASE-STATUS

CIVIL-ENGINEERING <P+> RUNWAY
                  <P+> PIPELINE
         PIPELINE <P+> FUEL-STORE

AIRBASE-STATUS <A> TASKING
               <A> FLYING-SCHEDULE
               <A> SORTIE-GENERATION

TASKING <P> MISSION-TYPE
        <P> TIME-FRAME
        <P> FLYING-SCHEDULE
        <P> SUPPORTING-AIRBASE

      FLYING-SCHEDULE <P>  SUPPORT-FOR-TASKING
                      <C>  SORTIE-GENERATION
  SUPPORT-FOR-TASKING <A+> MILITARY-ENVIRONMENT
           TIME-FRAME <A>  FLYING-SCHEDULE
```

Figure 4.

```
AIRCRAFT <A> FLYING-SCHEDULE

MECHANICAL-BREAKDOWN <A-> AIRCRAFT-STATUS
         MAINTENANCE <A+> AIRCRAFT-STATUS
                     <P+> AIRCRAFT
                     <C-> SPARE-PARTS
                     <C>  MAINTENANCE-CREW

SORTIE-GENERATION <C> AIRCREW
                  <C> MUNITIONS-LOADING
                  <C> AIRCRAFT

      THREAT <A> (SORTIE-GENERATION <C> MUNITIONS-LOADING)
     WEATHER <A> (SORTIE-GENERATION <C> MUNITIONS-LOADING)
      TARGET <A> (SORTIE-GENERATION <C> MUNITIONS-LOADING)
MISSION-TYPE <A> (SORTIE-GENERATION <C> MUNITIONS-LOADING)
             <A> (SORTIE-GENERATION <C> AIRCREW)
             <A> (SORTIE-GENERATION <C> AIRCRAFT)
AIRCRAFT-STATUS <A> (SORTIE-GENERATION <C> AIRCRAFT)


 MUNITIONS-LOADING <C> MUNITIONS-LOADING-CREW
                   <C> SCL
MUNITIONS-ASSEMBLY <C> MUNITIONS-STORE
                   <C> MUNITIONS-ASSEMBLY-CREW
                   <P> SCL
   MUNITIONS-STORE <A> MUNITIONS-ASSEMBLY              ; CUT SCL IN HALF
                   <A> FLYING-SCHEDULE                 ; CUT SORTIES
```

Figure 5.

```
SORTIE-GENERATION <P> BAI
                  <P> DCA
                  <P> ESCORT
                  <P> OCA
                  <P> RECCE
                  <P> SEAD

SORTIE-GENERATION <P>  RECOVERY

         RECOVERY <P+> AIRCREW
                  <P+> AIRCRAFT
                  <C>  REFUELING
        REFUELING <C-> FUEL-STORE
                  <C>  FUEL-TRUCK
                  <C>  REFUELING-CREW
         AIRCRAFT <A>  (REFUELING <C> FUEL-STORE)
```

Figure 6.

```
GROUND-TRANSPORT <P+> FUEL-STORE
                 <P+> MUNITIONS-STORE
                 <P+> SPARE-PARTS
         AIRLIFT <P+> MUNITIONS-STORE
                 <P+> SPARE-PARTS
             MAC <P+> AIRLIFT
            ARMY <P+> GROUND-TRANSPORT
         AIRLIFT <C>  REFUELING

DUTY-DAY <A> CREW-REST
         <A> FLYING-SCHEDULE

CREW-REST <P+> CREW

AIRCREW <C> TRAINING
```

```
ALERT-STATUS  <A>  AIRCRAFT
              <A>  DUTY-DAY
              <P+> CREW
```

7.3 Implied Relationships.

As mentioned in Section 7, the properties of the P/C/A network permit the system to work with implicit relationships based on the nature of production and consumption. One example of this is the maintenance status of Non Mission Capable (NMC) aircraft due to lack of maintenance crew (NMC/M) and lack of spare parts (NMC/S). The following two relationships explicitly state this:

```
MAINTENANCE-CREW [A] AIRCRAFT-STATUS            ; NMC/M
     SPARE-PARTS [A] AIRCRAFT-STATUS            ; NMC/S
```

However, as seen in Section 7.2, these two relations are implied by the production/consumption relation. Aircraft status is affected by maintenance, which in turn consumes (requires) spare parts and maintenance crew.

```
MAINTENANCE [A+] AIRCRAFT-STATUS
            [C]  MAINTENANCE-CREW
            [C]  SPARE-PARTS
```

A shortage of maintenance crew or spare parts, (consumed through some other mechanism) therefore, has a bad effect on the good effect that maintenance has on an aircraft's status. Thus, the network is able to infer the effect of NMC/M and MNC/S without explicit representation.

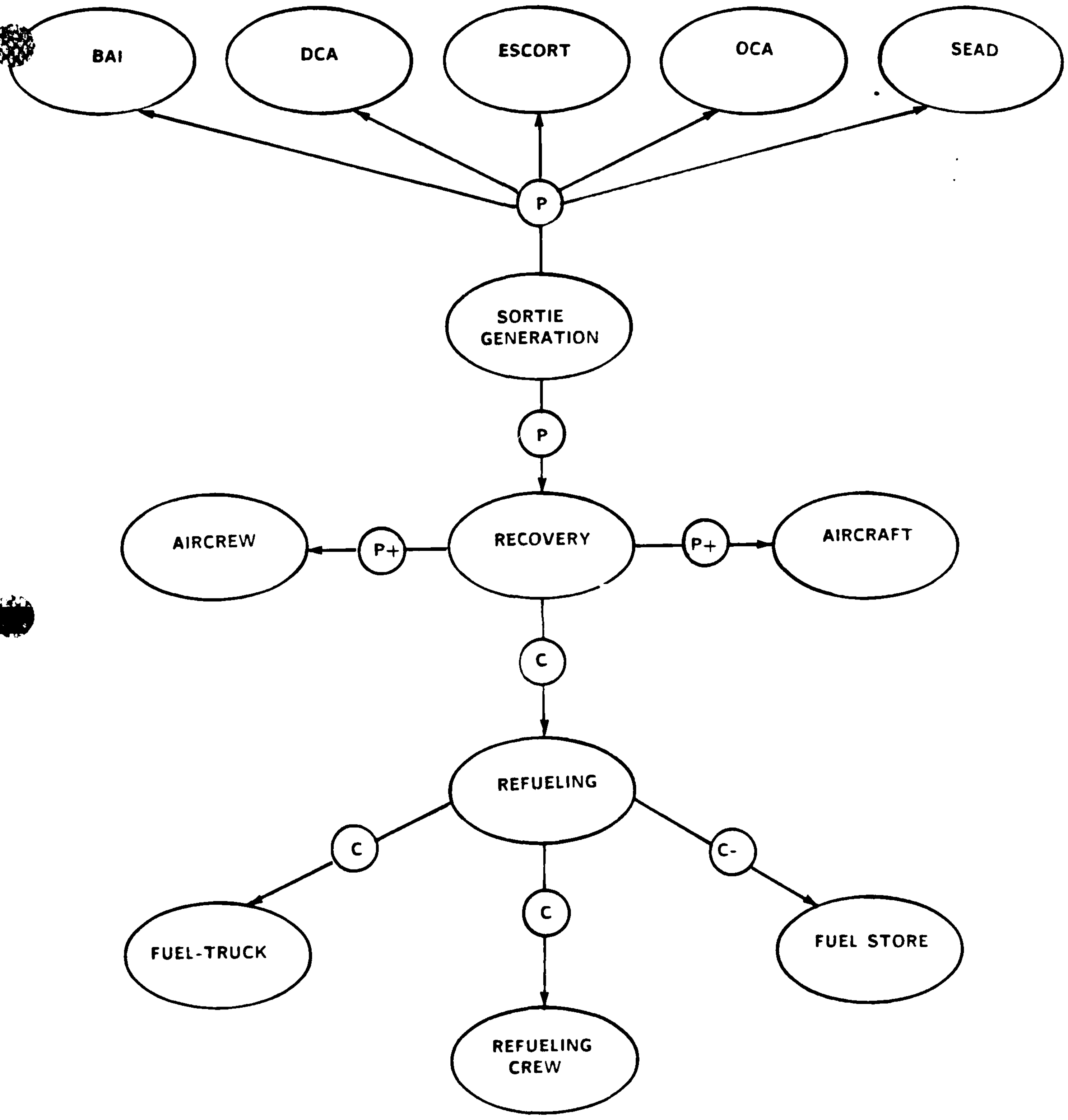
Another example is the representation that the fuel pipeline affects an airbase's status:

```
PIPELINE [A] AIRBASE-STATUS
```

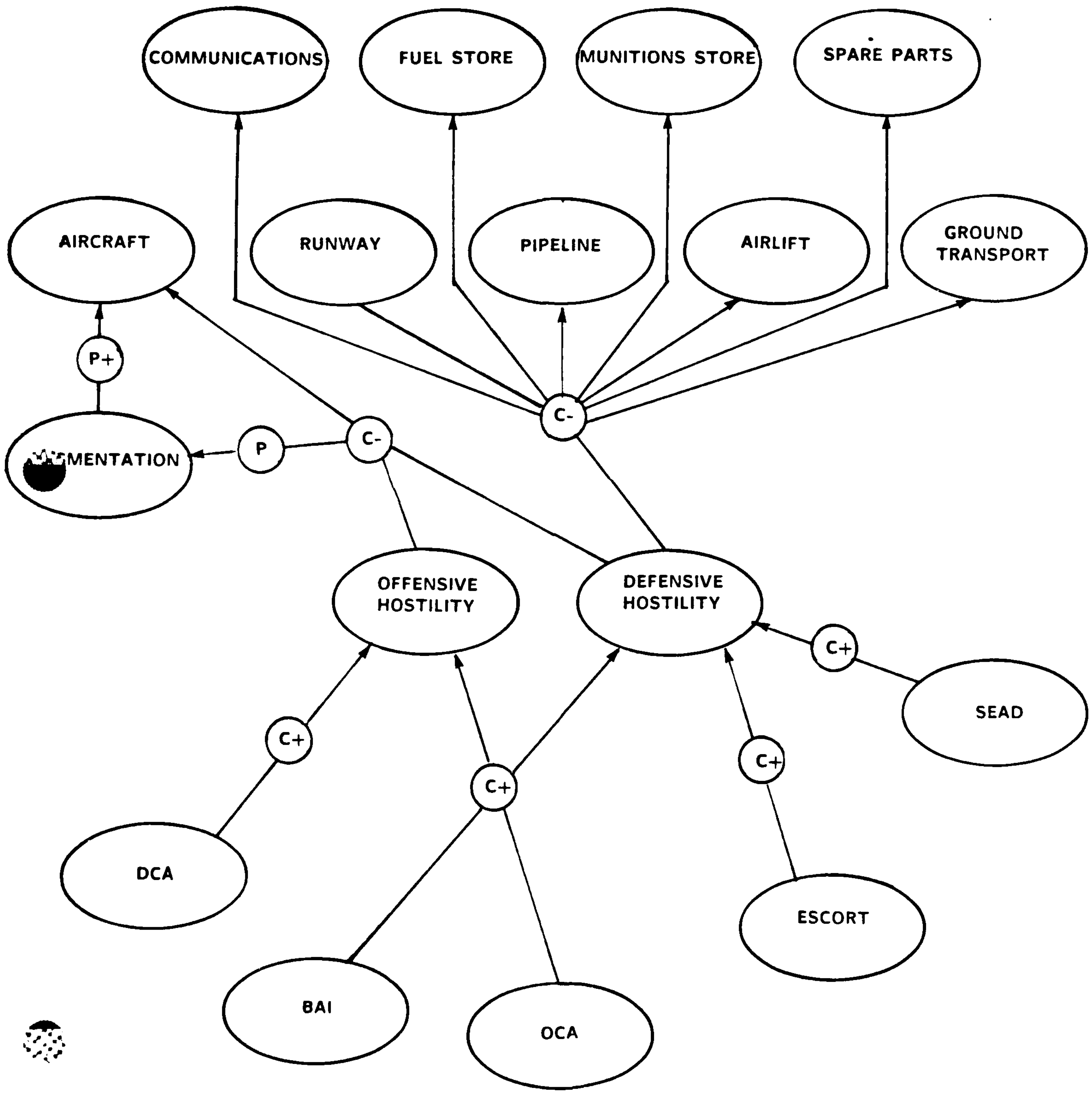
Since the fuel store affects the airbase status and the pipeline produces (resupplies) fuel:

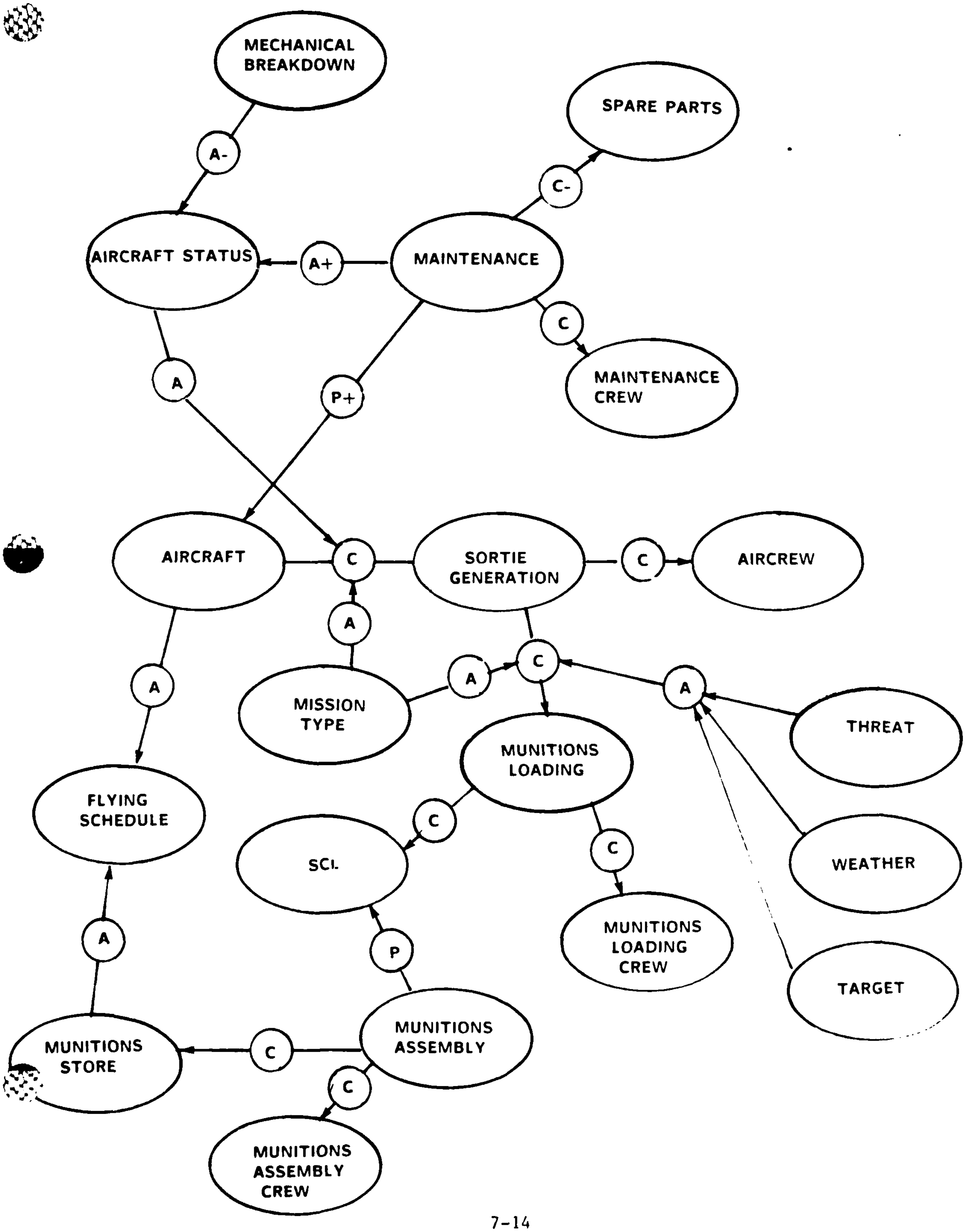
```
FUEL-STORE [A]  AIRBASE-STATUS
  PIPELINE [P+] FUEL
```

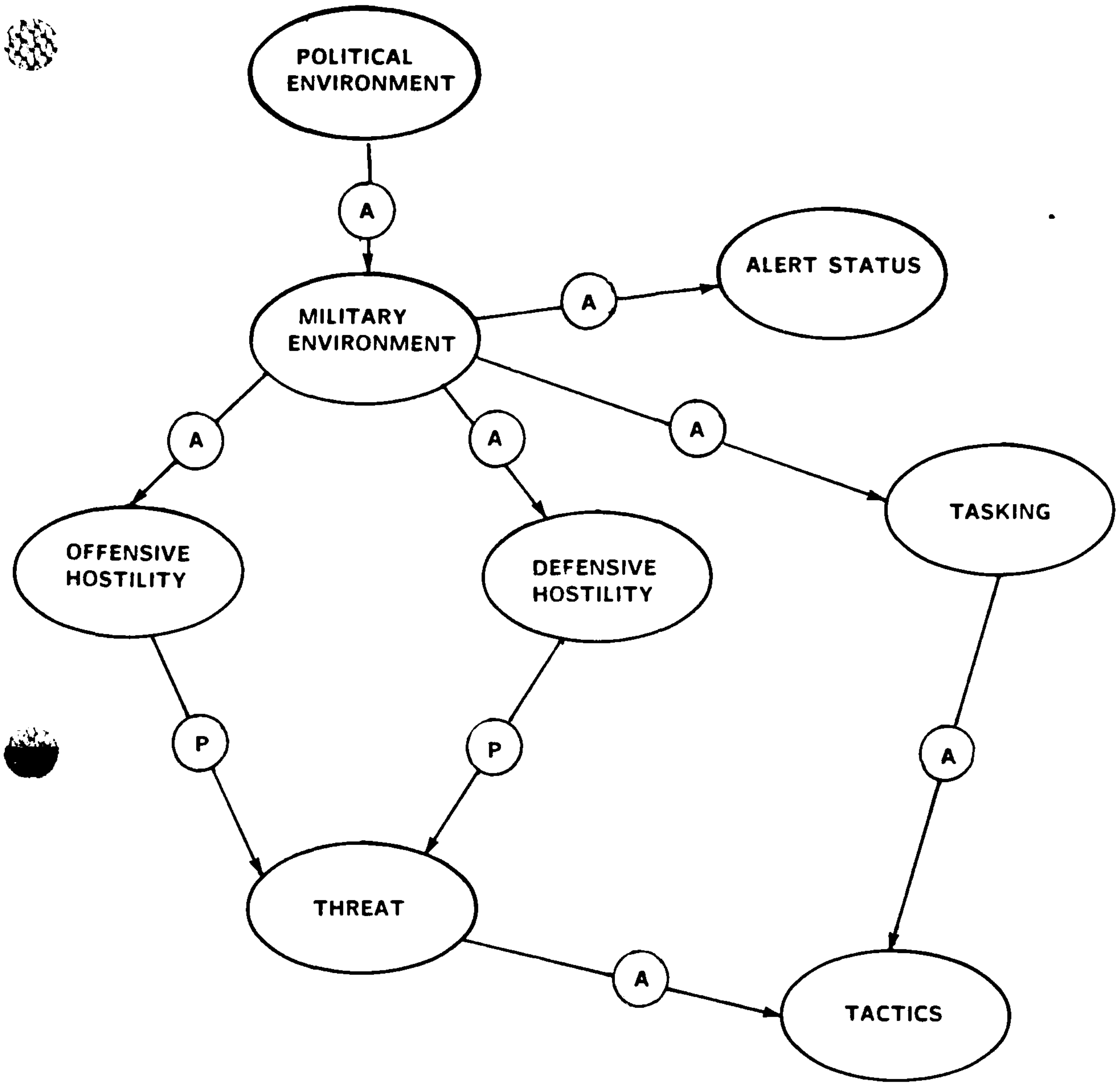
the network knows that the pipeline affects the status of an airbase.

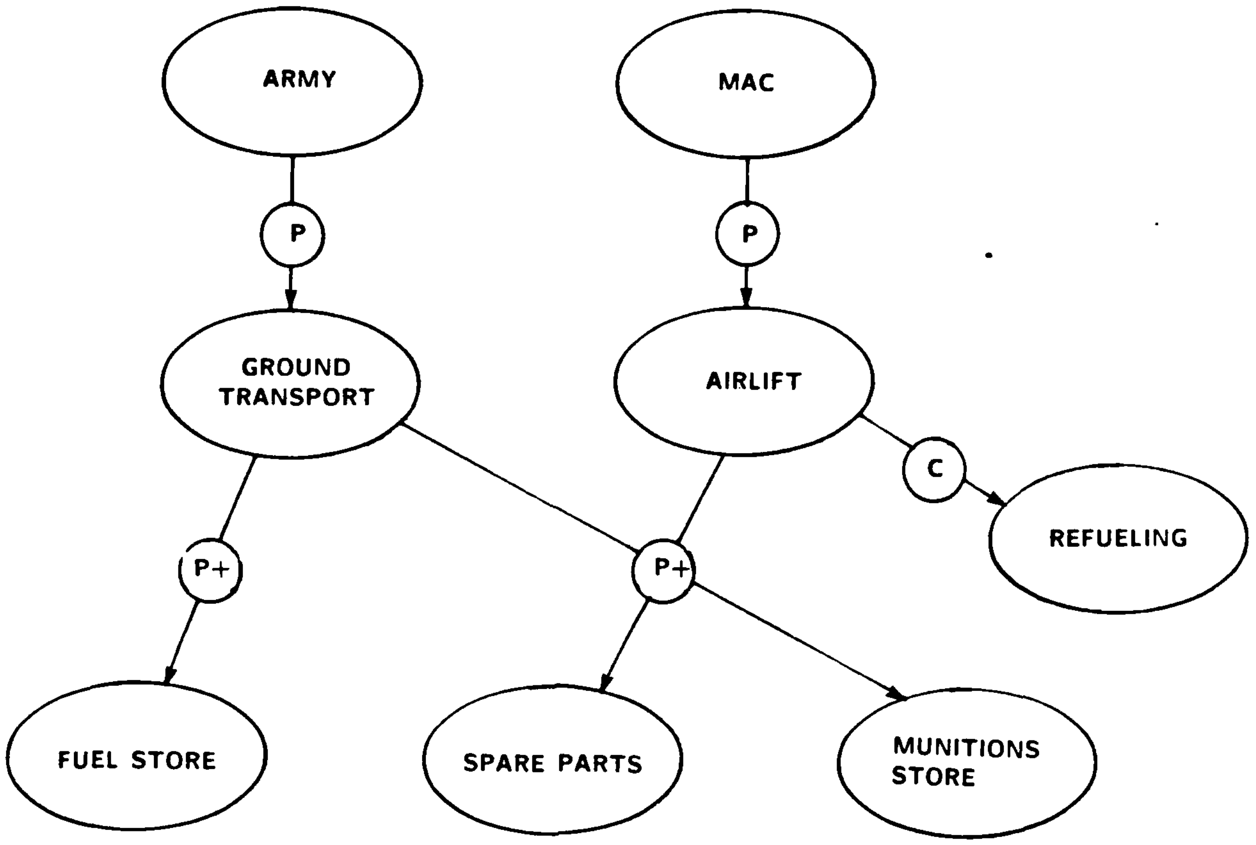

BAI
DCA
ESCORT
OCA
SEAD
P
SORTIE GENERATION
P
AIRCREW
P+
RECOVERY
P+
AIRCRAFT
C
REFUELING
C
C-
C
FUEL-TRUCK
FUEL STORE
REFUELING CREW

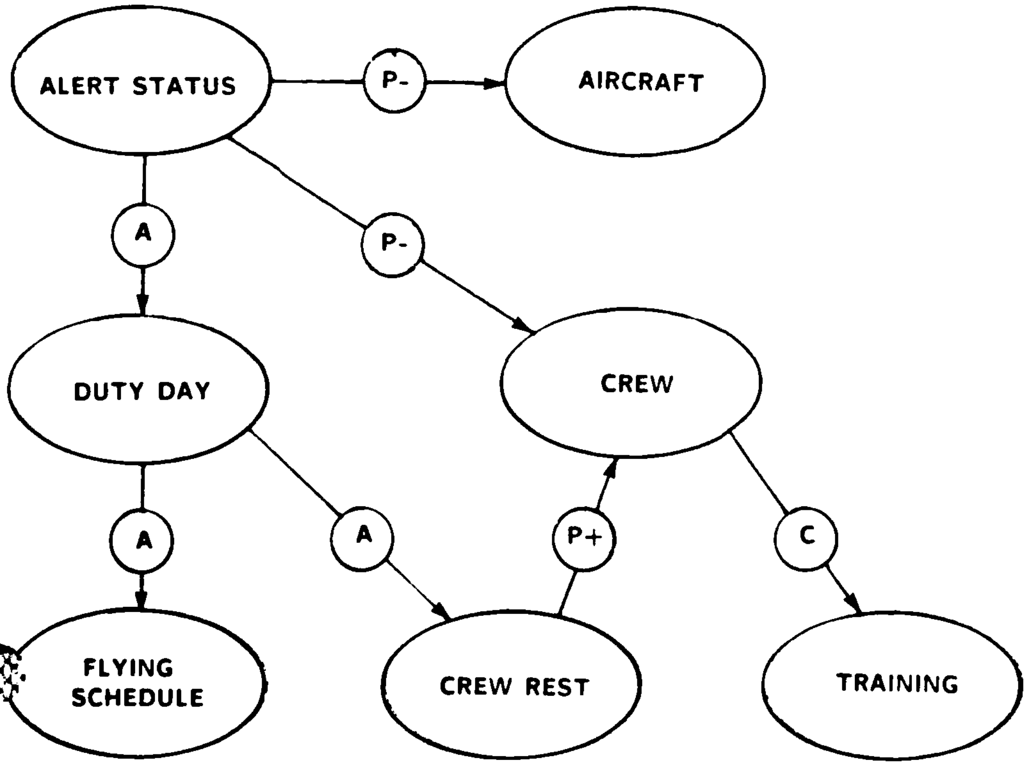
Figures

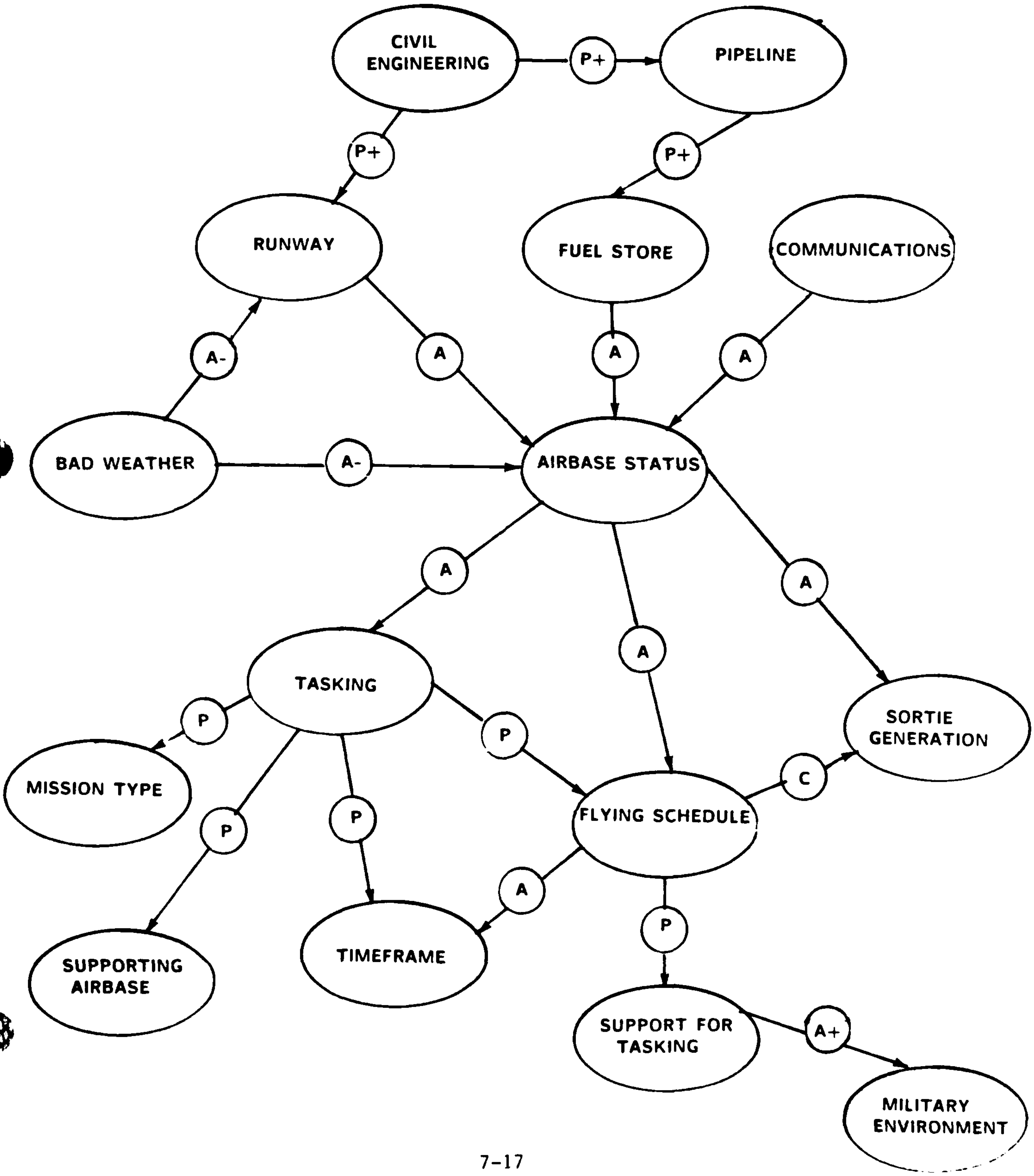

COMMUNICATIONS
FUEL STORE
MUNITIONS STORE
SPARE PARTS
AIRCRAFT
RUNWAY
PIPELINE
AIRLIFT
GROUND TRANSPORT
P+
C-
MENTATION
P
C-
OFFENSIVE HOSTILITY
DEFENSIVE HOSTILITY
C+
SEAD
C+
C+
C+
DCA
ESCORT
BAI
OCA


MECHANICAL BREAKDOWN
A-
SPARE PARTS
C-
AIRCRAFT STATUS
A+
MAINTENANCE
C
MAINTENANCE CREW
A
P+
AIRCRAFT
C
SORTIE GENERATION
C
AIRCREW
A
C
A
A
A
MISSION TYPE
THREAT
MUNITIONS LOADING
FLYING SCHEDULE
C
SCL
C
WEATHER
MUNITIONS LOADING CREW
A
P
TARGET
MUNITIONS ASSEMBLY
MUNITIONS STORE
C
C
MUNITIONS ASSEMBLY CREW


POLITICAL ENVIRONMENT
A
MILITARY ENVIRONMENT
A
ALERT STATUS
A
A
A
OFFENSIVE HOSTILITY
DEFENSIVE HOSTILITY
TASKING
P
P
A
THREAT
A
TACTICS


ARMY
MAC
P
P
GROUND TRANSPORT
AIRLIFT
C
REFUELING
P+
P+
FUEL STORE
SPARE PARTS
MUNITIONS STORE


ALERT STATUS
P-
AIRCRAFT
A
P-
DUTY DAY
CREW
A
A
P+
C
FLYING SCHEDULE
CREW REST
TRAINING


CIVIL ENGINEERING
P+
PIPELINE
P+
P+
RUNWAY
FUEL STORE
COMMUNICATIONS
A-
A
A
A
BAD WEATHER
A-
AIRBASE STATUS
A
A
A
TASKING
SORTIE GENERATION
P
P
MISSION TYPE
C
P
P
FLYING SCHEDULE
A
P
SUPPORTING AIRBASE
TIMEFRAME
SUPPORT FOR TASKING
A+
MILITARY ENVIRONMENT

VIII. Conclusions

There are important differences between designing and building complex data base management information systems and expert systems. Expert systems approach problems the way people do. The challenges in designing and building such systems lie less in the structure of files, the type of variables, the relationship between variables, etc., than in discovering the logic, knowledge, and reasoning people use to solve problems. Because reasoning and knowledge are so complicated, expert systems cannot be designed on paper. The best design approach is to build an experimental prototype that allows the designer to test their knowledge and logic. This is the approach we have used.